ESEIAAT

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Study: models of sounding balloon trajectory

Report Attachment

Author: Oriol Alís González

Director: Dr. Manel Soria Guerrero

30/09/2019

Master's degree in Aerospace Engineering

Index of contents

Index of figures

Index of tables

Annex A

In this Annex, several materials and studies that complement the Master Thesis Report are presented.

A.1. GFS available data

In this section the complete available data from GFS model is listed in Table 1:

Name	*Description*	*Units*
LatLon_Projection		
lat		**degrees_north**
lon		**degrees_east**
reftime	GRIB reference time	Hour since 2017-12-31T00:00:00Z
time	GRIB forecast or observation time	Hour since 2017-12-31T00:00:00Z
height_above_ground	Specified height level above ground	m
pressure_difference_layer	Level at specified pressure difference from ground to level	Pa
pressure_difference_layer_bounds	bounds for pressure_difference_layer	Pa
altitude_above_msl	Specific altitude above mean sea level	m
isobaric	Isobaric surface	Pa
height_above_ground_layer	Specified height level above ground	m
height_above_ground_layer_bounds	bounds for height_above_ground_layer	m
height_above_ground_layer1	Specified height level above ground	m
height_above_ground_layer1_bounds	bounds for height_above_ground_layer1	m
isobaric1	Isobaric surface	Pa
isobaric2	**Isobaric surface**	**Pa**
pressure_difference_layer1	Level at specified pressure difference from ground to level	Pa
pressure_difference_layer1_bounds	bounds for pressure_difference_layer1	Pa
isobaric3	Isobaric surface	Pa
sigma_layer	Sigma level	sigma
sigma_layer_bounds	bounds for sigma_layer	sigma
depth_below_surface_layer	Depth below land surface	m
depth_below_surface_layer_bounds	bounds for depth_below_surface_layer	m
height_above_ground1	Specified height level above ground	m
height_above_ground2	Specified height level above ground	m
height_above_ground3	Specified height level above ground	m
potential_vorticity_surface	Potential vorticity surface	K m2 kg-1 s-1
pressure_difference_layer2	Level at specified pressure difference from ground to level	Pa
pressure_difference_layer2_bounds	bounds for pressure_difference_layer2	Pa
isobaric4	**Isobaric surface**	**Pa**
sigma	Sigma level	sigma
height_above_ground4	Specified height level above ground	m
Absolute_vorticity_isobaric	Absolute vorticity @ Isobaric surface	1/s

Apparent_temperature_height_above_ground	Apparent temperature @ Specified height level above	K
Cloud_mixing_ratio_isobaric	Cloud mixing ratio @ Isobaric surface	kg/kg
Cloud_water_entire_atmosphere_single_layer	Cloud water @ Entire atmosphere layer	kg.m-2
Convective_available_potential_energy_surface	Convective available potential energy @ Ground or water	J/kg
Convective_available_potential_energy_pressure_difference_layer	Convective available potential energy @ Level at specified	J/kg
Convective_inhibition_surface	Convective inhibition @ Ground or water surface	J/kg
Convective_inhibition_pressure_difference_layer	Convective inhibition @ Level at specified pressure difference	J/kg
Dewpoint_temperature_height_above_ground	Dewpoint temperature @ Specified height level above	K
Geopotential_height_surface	Geopotential height @ Ground or water surface	gpm
Geopotential_height_isobaric	Geopotential height @ Isobaric surface	gpm
Geopotential_height_zeroDegC_isotherm	Geopotential height @ Level of 0 °C isotherm	gpm
Geopotential_height_potential_vorticity_surface	Geopotential height @ Potential vorticity surface	gpm
Geopotential_height_maximum_wind	Geopotential height @ Maximum wind level	gpm
Geopotential_height_tropopause	Geopotential height @ Tropopause	gpm
Geopotential_height_highest_tropospheric_freezing	Geopotential height @ Highest tropospheric freezing level	gpm
Haines_index_surface	Haines index @ Ground or water surface	
ICAO_Standard_Atmosphere_Reference_Height_maximum_wind	ICAO Standard Atmosphere Reference Height @ Maximum	m
ICAO_Standard_Atmosphere_Reference_Height_tropopause	ICAO Standard Atmosphere Reference Height @	m
Ice_cover_surface	Ice cover @ Ground or water surface	
Land_cover_0__sea_1__land_surface	Land cover (0 = sea, 1 = land) @ Ground or water surface	
Per_cent_frozen_precipitation_surface	Per cent frozen precipitation @ Ground or water surface	%
Potential_temperature_sigma	Potential temperature @ Sigma level	K
Precipitable_water_entire_atmosphere_single_layer	Precipitable water @ Entire atmosphere layer	kg.m-2
Pressure_surface	Pressure @ Ground or water surface	Pa
Pressure_potential_vorticity_surface	Pressure @ Potential vorticity surface	Pa
Pressure_maximum_wind	Pressure @ Maximum wind level	Pa
Pressure_tropopause	Pressure @ Tropopause	Pa
Pressure_height_above_ground	Pressure @ Specified height level above ground	Pa
Pressure_reduced_to_MSL_msl	Pressure reduced to MSL @ Mean sea level	Pa
Relative_humidity_pressure_difference_layer	Relative humidity @ Level at specified pressure difference	%
Relative_humidity_sigma_layer	Relative humidity @ Sigma level layer	%
Relative_humidity_isobaric	Relative humidity @ Isobaric surface	%
Relative_humidity_zeroDegC_isotherm	Relative humidity @ Level of 0 °C isotherm	%
Relative_humidity_height_above_ground	Relative humidity @ Specified height level above ground	%
Relative_humidity_entire_atmosphere_single_layer	Relative humidity @ Entire atmosphere layer	%
Relative_humidity_highest_tropospheric_freezing	Relative humidity @ Highest tropospheric freezing level	%
Relative_humidity_sigma	Relative humidity @ Sigma level	%
Snow_depth_surface	Snow depth @ Ground or water surface	m
Soil_temperature_depth_below_surface_layer	Soil temperature @ Depth below land surface layer	K
Specific_humidity_pressure_difference_layer	Specific humidity @ Level at specified pressure difference	kg/kg

Specific_humidity_height_above_ground	Specific humidity @ Specified height level above ground	kg/kg
Storm_relative_helicity_height_above_ground_layer	Storm relative helicity @ Specified height level above	J/kg
Temperature_surface	Temperature @ Ground or water surface	K
Temperature_pressure_difference_layer	Temperature @ Level at specified pressure difference	K
Temperature_isobaric	Temperature @ Isobaric surface	K
Temperature_potential_vorticity_surface	Temperature @ Potential vorticity surface	K
Temperature_maximum_wind	Temperature @ Maximum wind level	K
Temperature_altitude_above_msl	Temperature @ Specific altitude above mean sea level	K
Temperature_height_above_ground	Temperature @ Specified height level above ground	K
Temperature_tropopause	Temperature @ Tropopause	K
Temperature_sigma	Temperature @ Sigma level	K
Total_ozone_entire_atmosphere_single_layer	Total ozone @ Entire atmosphere layer	DU
Ozone_Mixing_Ratio_isobaric	Ozone Mixing Ratio @ Isobaric surface	kg.kg-1
Vertical_Speed_Shear_tropopause	Vertical Speed Shear @ Tropopause	s-1
Vertical_Speed_Shear_potential_vorticity_surface	Vertical Speed Shear @ Potential vorticity surface	s-1
U-Component_Storm_Motion_height_above_ground_layer	U-Component Storm Motion @ Specified height level above	m.s-1
V-Component_Storm_Motion_height_above_ground_layer	V-Component Storm Motion @ Specified height level above	m.s-1
Ventilation_Rate_planetary_boundary	Ventilation Rate @ Planetary Boundary Layer	m2.s-1
MSLP_Eta_model_reduction_msl	MSLP (Eta model reduction) @ Mean sea level	Pa
5-Wave_Geopotential_Height_isobaric	5-Wave Geopotential Height @ Isobaric surface	gpm
Planetary_Boundary_Layer_Height_surface	Planetary Boundary Layer Height @ Ground or water	m
Pressure_of_level_from_which_parcel_was_lifted_pressure difference_layer	Pressure of level from which parcel was lifted @ Level at	Pa
Sunshine_Duration_surface	Sunshine Duration @ Ground or water surface	s
Surface_Lifted_Index_surface	Surface Lifted Index @ Ground or water surface	K
Best_4_layer_Lifted_Index_surface	Best (4 layer) Lifted Index @ Ground or water surface	K
Volumetric_Soil_Moisture_Content_depth_below_surface_layer	Volumetric Soil Moisture Content @ Depth below land	Fraction
Wilting_Point_surface	Wilting Point @ Ground or water surface	Fraction
Land-sea_coverage_nearest_neighbor_land1sea0_surface	Land-sea coverage (nearest neighbor) [land=1,sea=0] @	
Field_Capacity_surface	Field Capacity @ Ground or water surface	Fraction
Vertical_velocity_pressure_isobaric	**Vertical velocity (pressure) @ Isobaric surface**	**Pa/s**
Vertical_velocity_pressure_sigma	Vertical velocity (pressure) @ Sigma level	Pa/s
Visibility_surface	Visibility @ Ground or water surface	m
Water_equivalent_of_accumulated_snow_depth_surface	Water equivalent of accumulated snow depth @	kg.m-2
Wind_speed_gust_surface	Wind speed (gust) @ Ground or water surface	m/s
u-component_of_wind_pressure_difference_layer	u-component of wind @ Level at specified pressure difference	m/s
u-component_of_wind_planetary_boundary	u-component of wind @ Planetary Boundary Layer	m/s
u-component_of_wind_isobaric	**u-component of wind @ Isobaric surface**	**m/s**
u-component_of_wind_potential_vorticity_surface	u-component of wind @ Potential vorticity surface	m/s
u-component_of_wind_maximum_wind	u-component of wind @ Maximum wind level	m/s
u-component_of_wind_altitude_above_msl	u-component of wind @ Specific altitude above mean	m/s

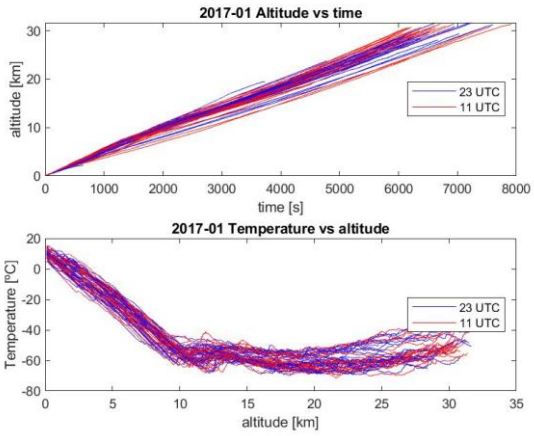
u-component_of_wind_height_above_ground	u-component of wind @ Specified height level above	m/s
u-component_of_wind_tropopause	u-component of wind @ Tropopause	m/s
u-component_of_wind_sigma	u-component of wind @ Sigma level	m/s
v-component_of_wind_pressure_difference_layer	v-component of wind @ Level at specified pressure difference	m/s
v-component_of_wind_planetary_boundary	v-component of wind @ Planetary Boundary Layer	m/s
v-component_of_wind_isobaric	**v-component of wind @ Isobaric surface**	**m/s**
v-component_of_wind_potential_vorticity_surface	v-component of wind @ Potential vorticity surface	m/s
v-component_of_wind_maximum_wind	v-component of wind @ Maximum wind level	m/s
v-component_of_wind_altitude_above_msl	v-component of wind @ Specific altitude above mean	m/s
v-component_of_wind_height_above_ground	v-component of wind @ Specified height level above	m/s
v-component_of_wind_tropopause	v-component of wind @ Tropopause	m/s
v-component_of_wind_sigma	v-component of wind @ Sigma level	m/s

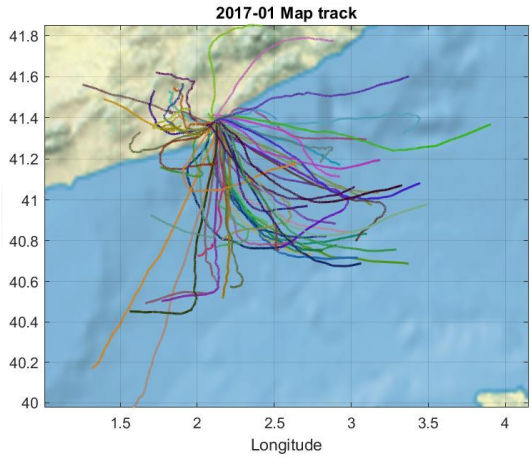
Table 1 GFS available data

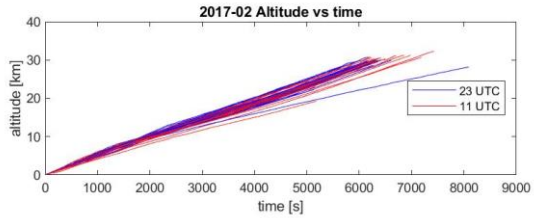
A.2. Radiosonde data study

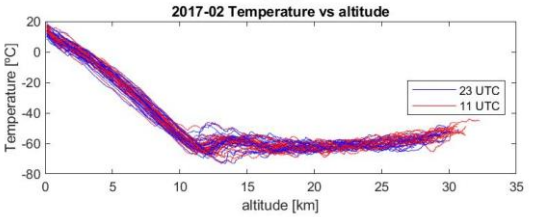
In order to understand the information available from the radiosonde, the variation of altitude with time and the temperature has been studied using script Main_Data_Study. The plots for years 2013-2017 have been represented for each month, with the blue line being the night launches while the red curves are the day ones. In the following figures this ascension profile as well as the trajectory is shown for every year and month:

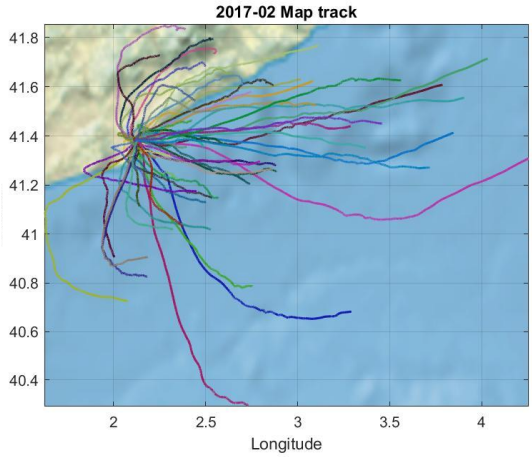
A.2.1. Year 2017 study

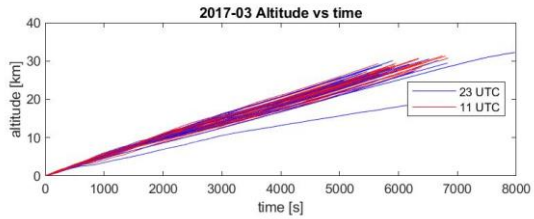


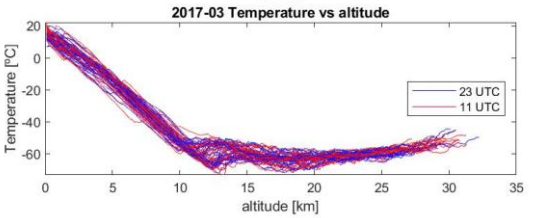


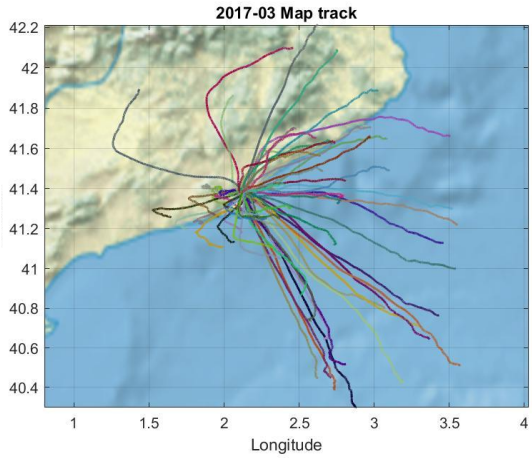


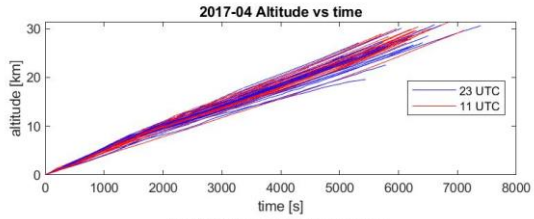


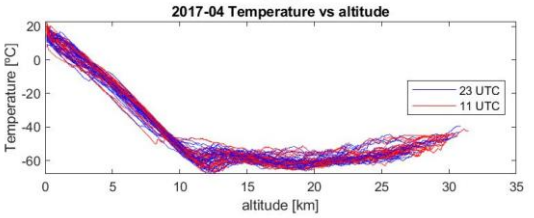


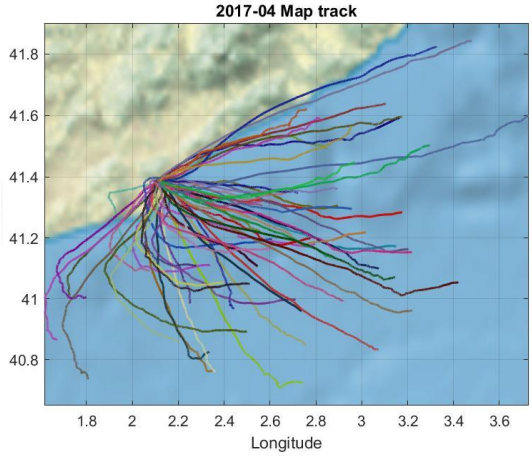


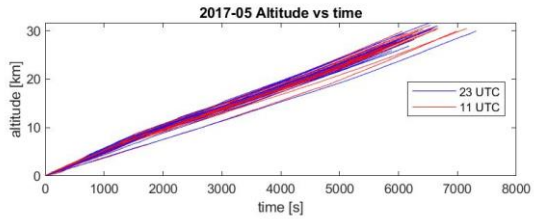


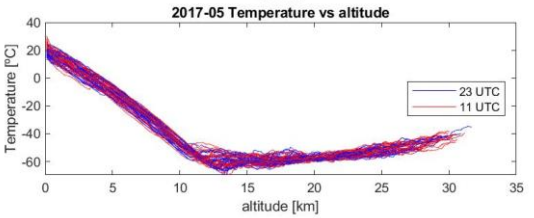


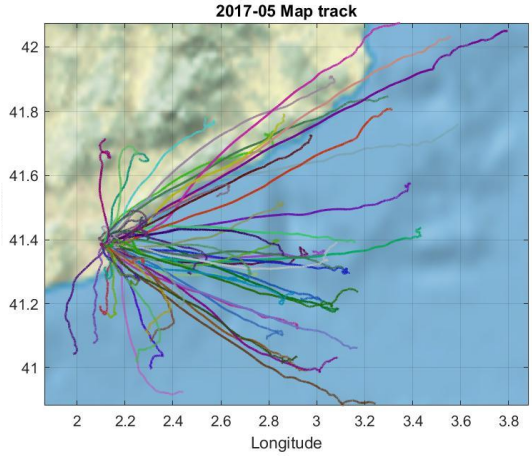

2017-04 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

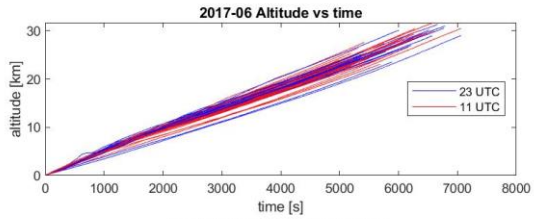

2017-04 Temperature vs altitude
Temperature [ºC]
altitude [km]
23 UTC
11 UTC

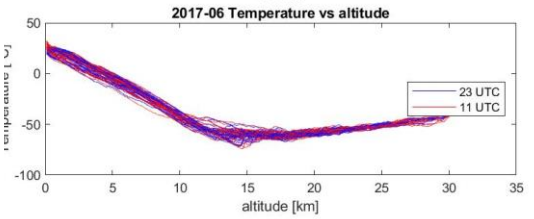

2017-04 Map track
Longitude

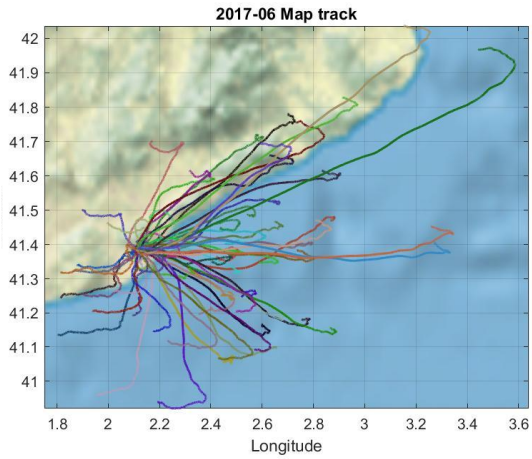

2017-05 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

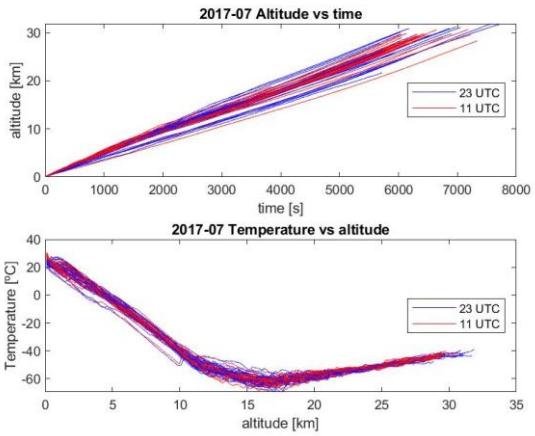

2017-05 Temperature vs altitude
Temperature [ºC]
altitude [km]
23 UTC
11 UTC


2017-05 Map track
Longitude

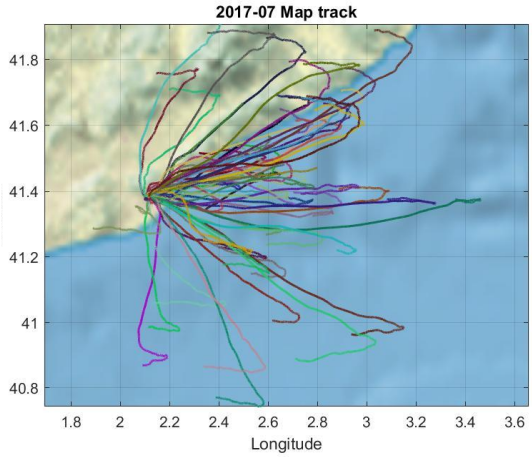

2017-06 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

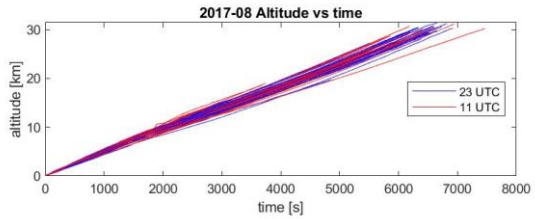

2017-06 Temperature vs altitude
altitude [km]
23 UTC
11 UTC

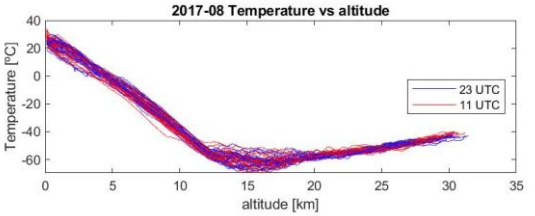

2017-06 Map track
Longitude

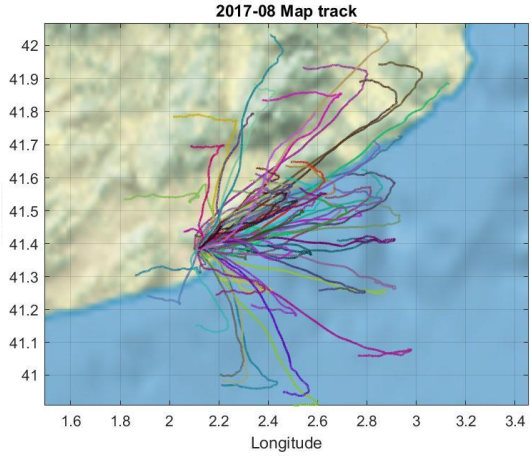

2017-07 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

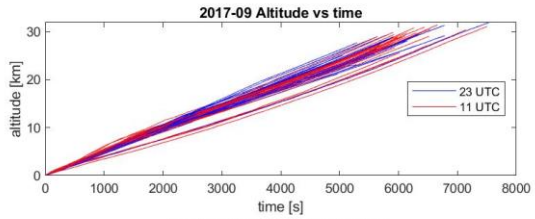
2017-07 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

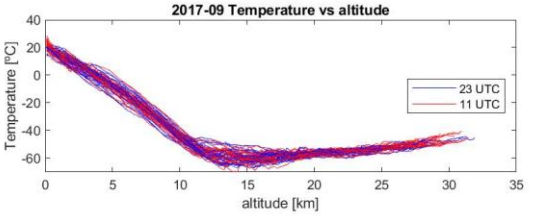

2017-07 Map track
Longitude

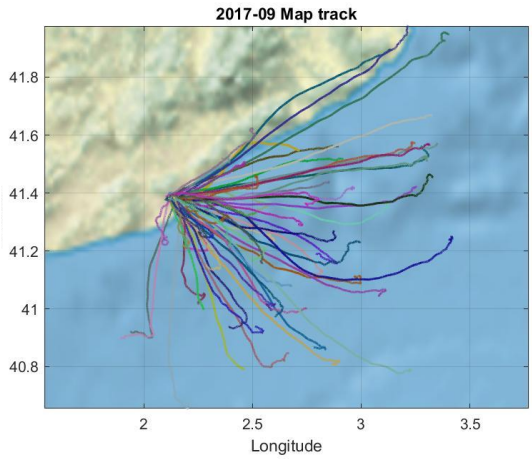

2017-08 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC


2017-08 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC


2017-08 Map track
Longitude


2017-09 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC


2017-09 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC


2017-09 Map track
Longitude

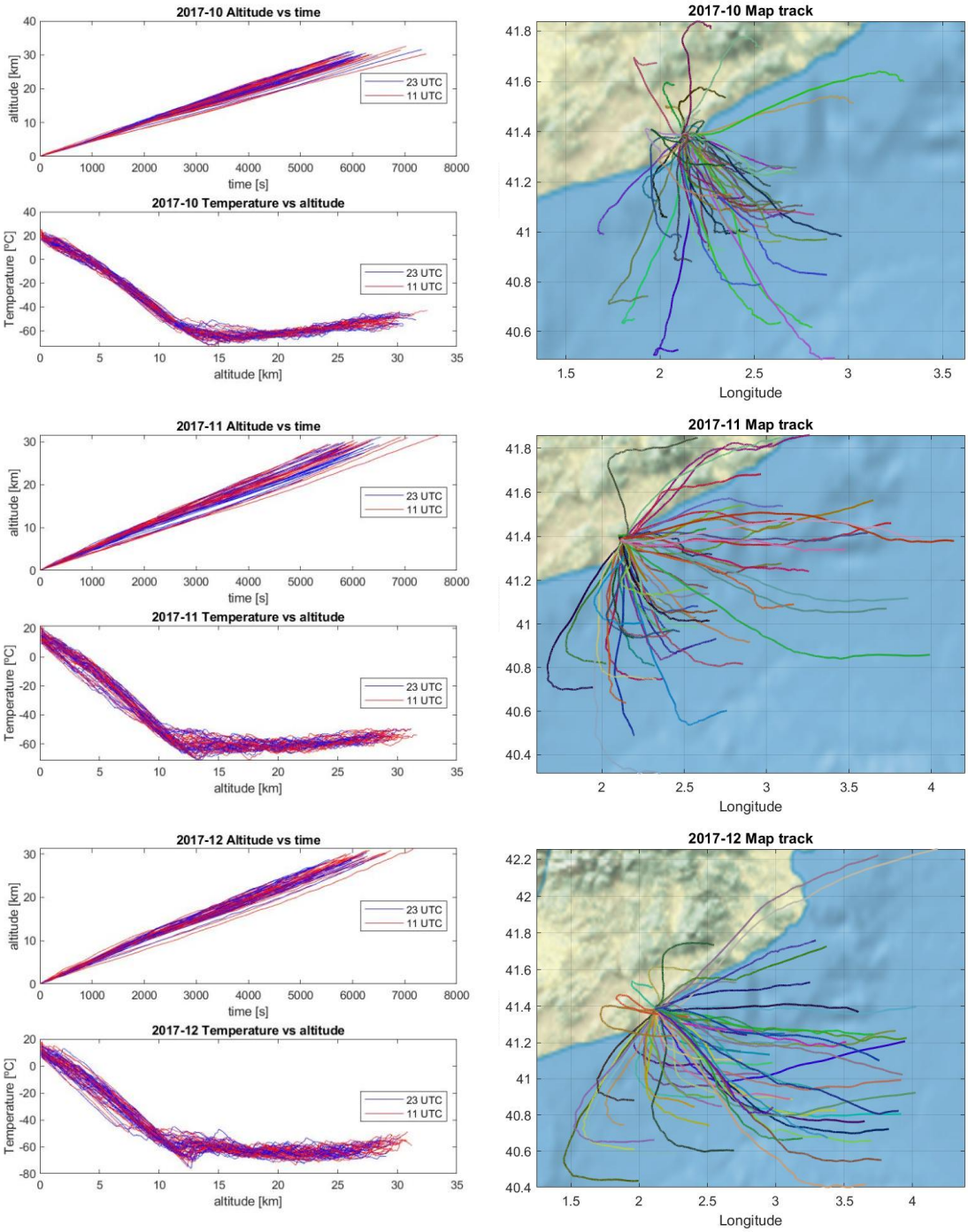
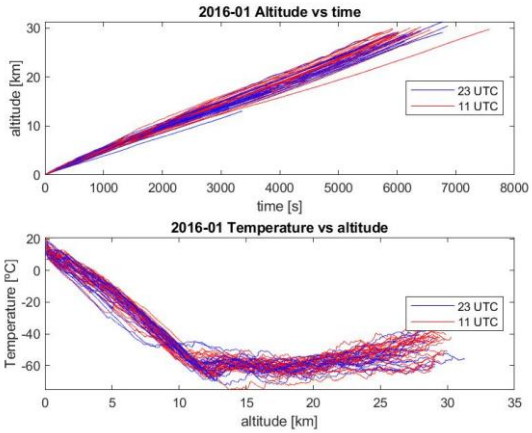
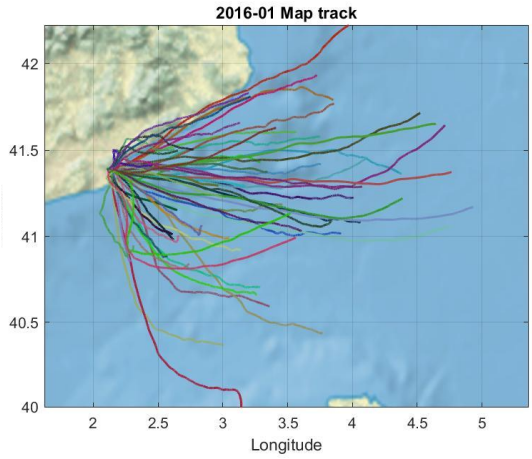
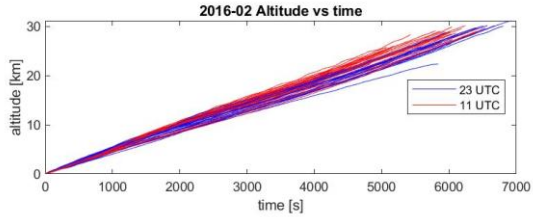


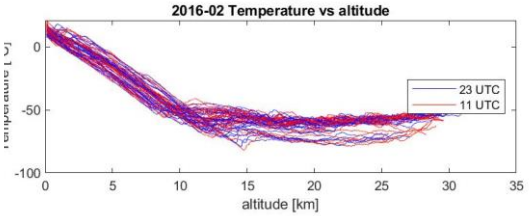
Figure 1 2017 Month by month altitude vs time and temperature vs altitude and flight trajectory

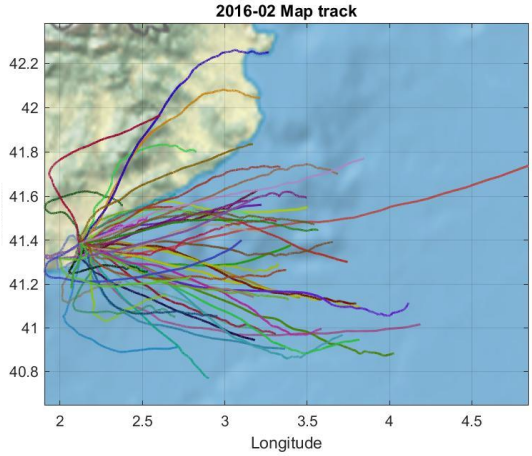
A.2.2. Year 2016 study

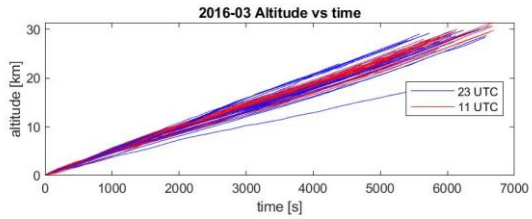


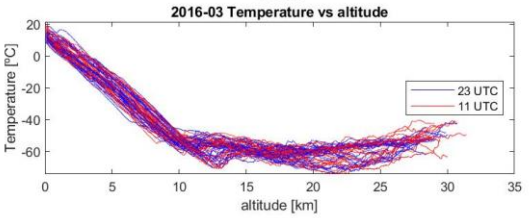


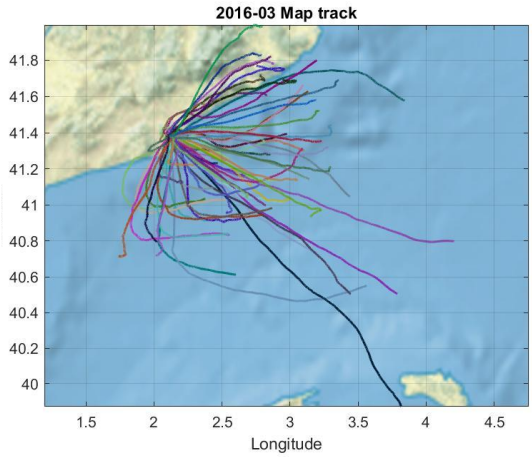


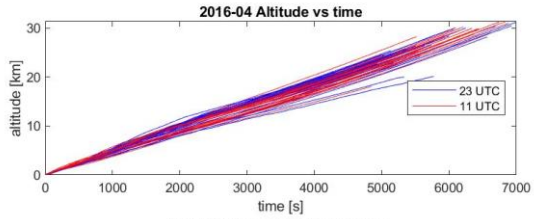


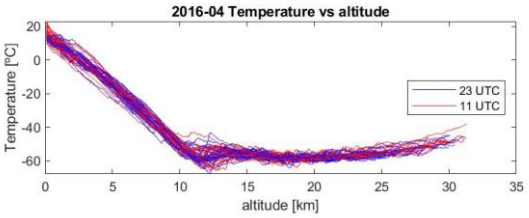


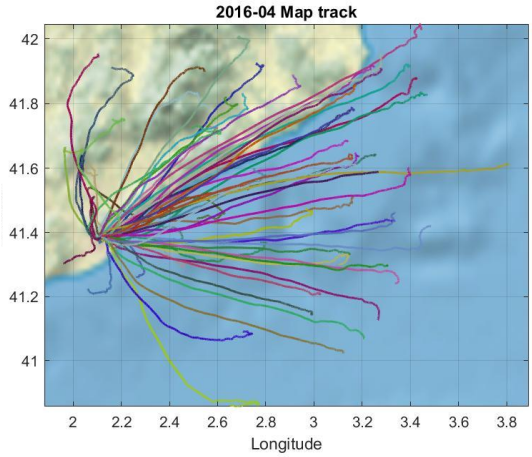


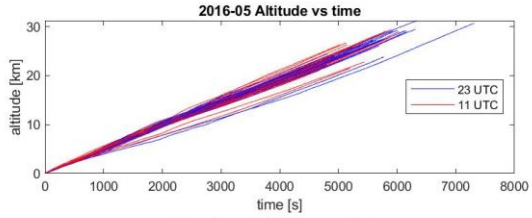


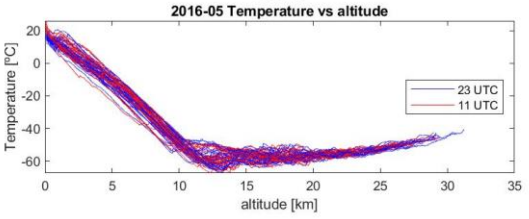


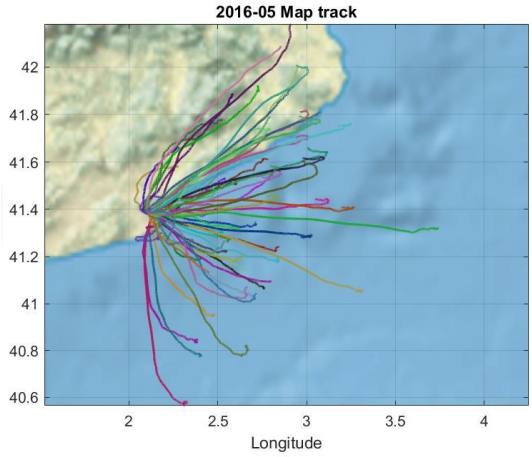

2016-04 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

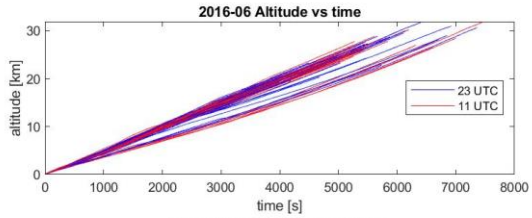

2016-04 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

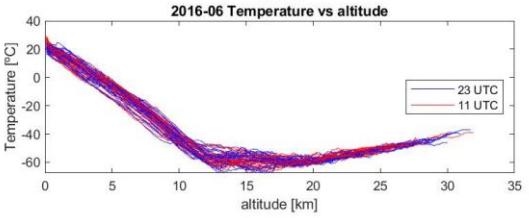

2016-04 Map track
Longitude

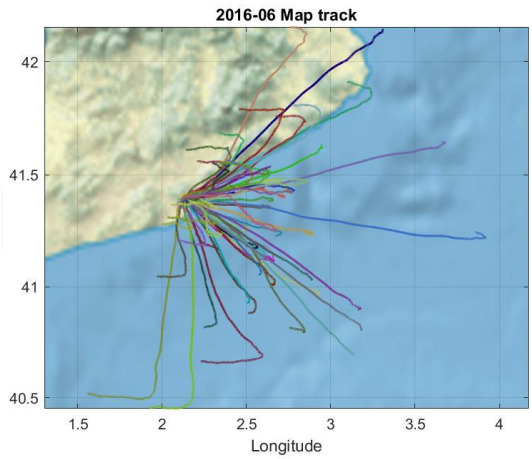

2016-05 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

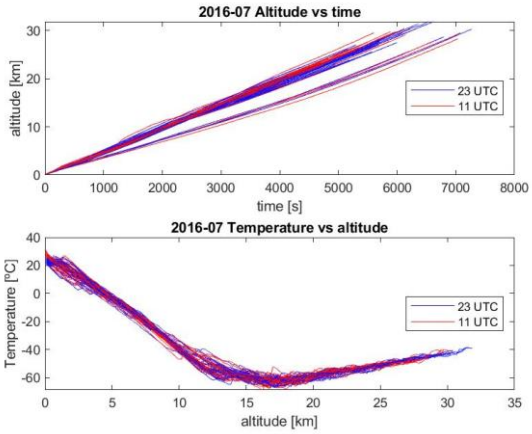

2016-05 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

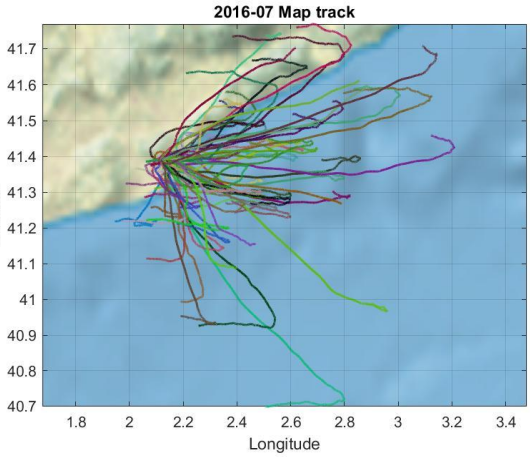

2016-05 Map track
Longitude

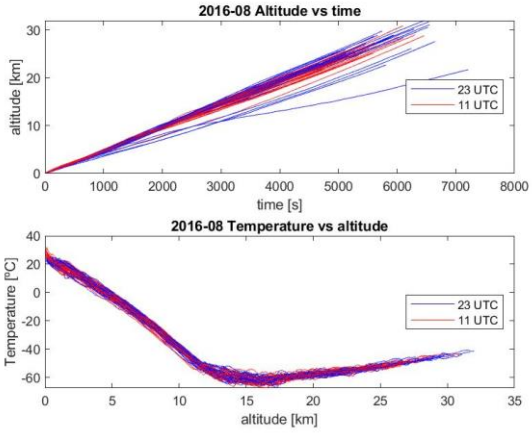

2016-06 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

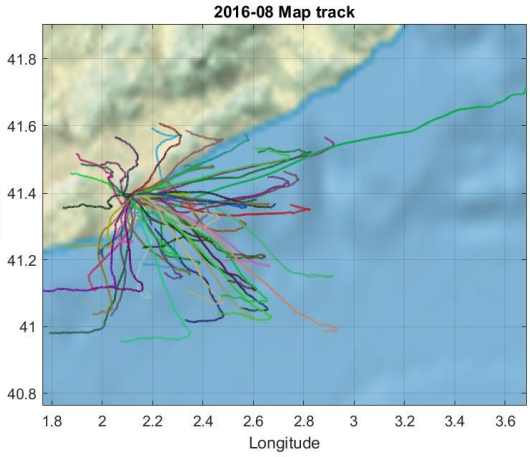

2016-06 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

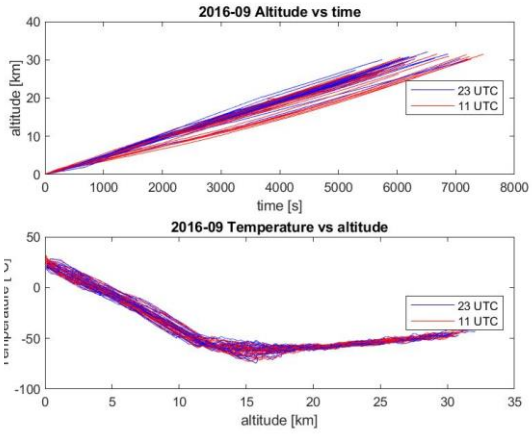

2016-06 Map track
Longitude

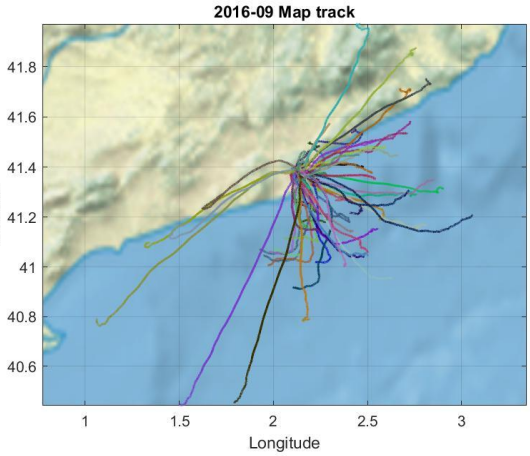

2016-07 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC
2016-07 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC


2016-07 Map track
Longitude


2016-08 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC
2016-08 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC


2016-08 Map track
Longitude


2016-09 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC
2016-09 Temperature vs altitude
altitude [km]
23 UTC
11 UTC


2016-09 Map track
Longitude

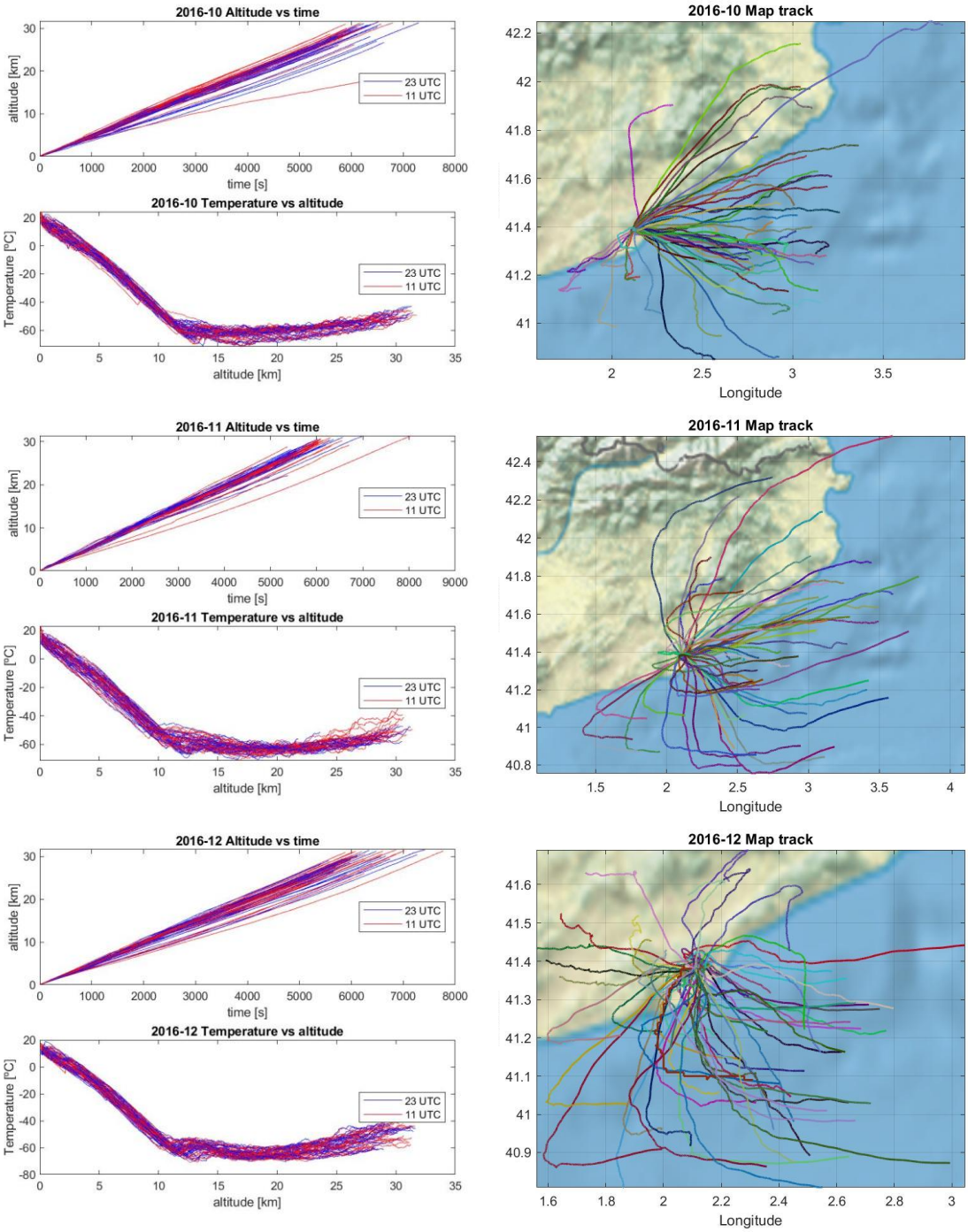
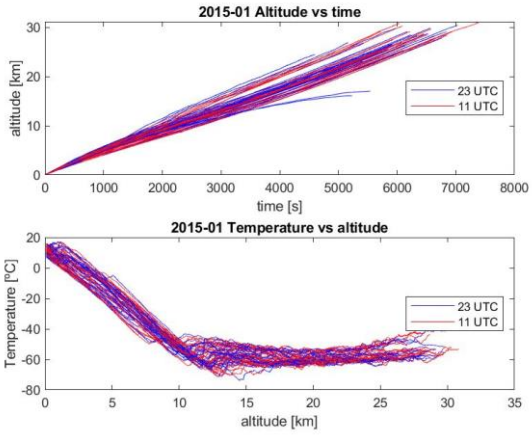
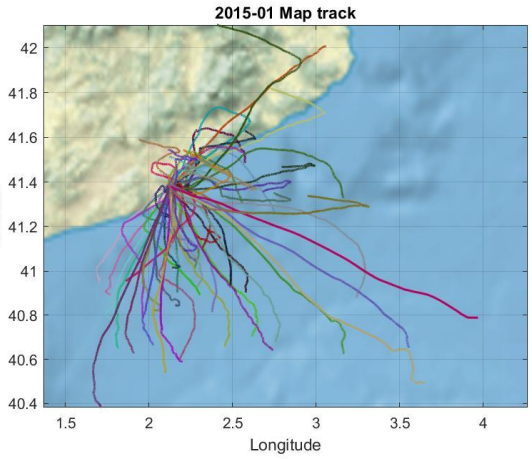
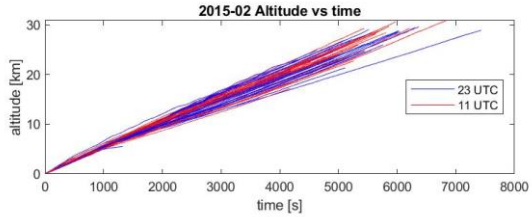


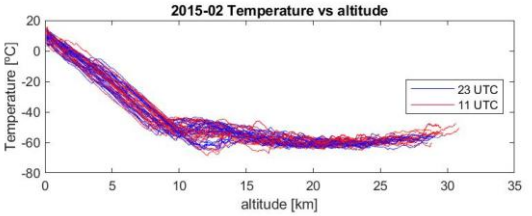
Figure 2 2016 Month by month altitude vs time and temperature vs altitude and flight trajectory

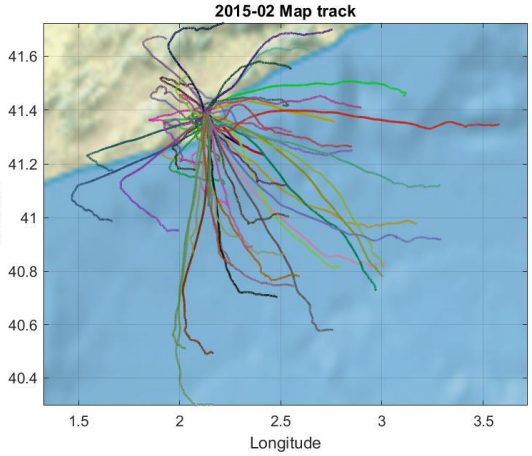
A.2.3. Year 2015 study

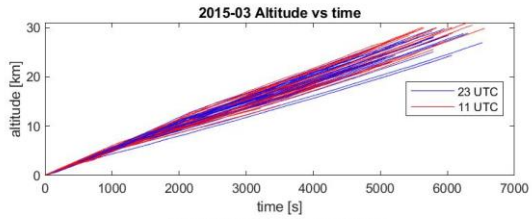


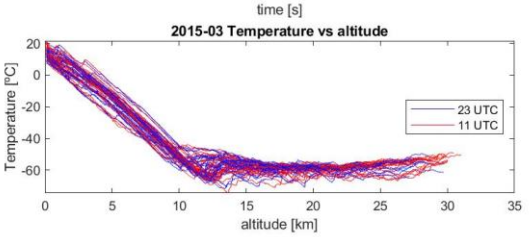


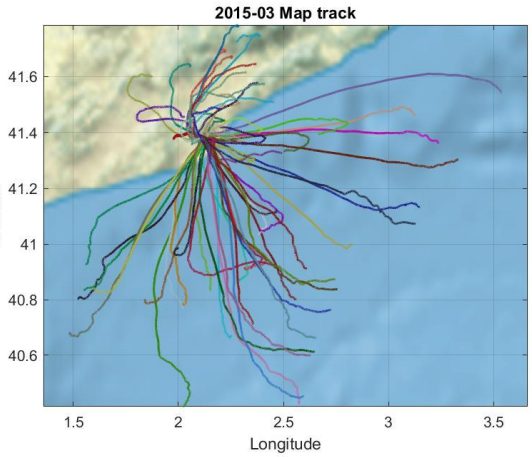


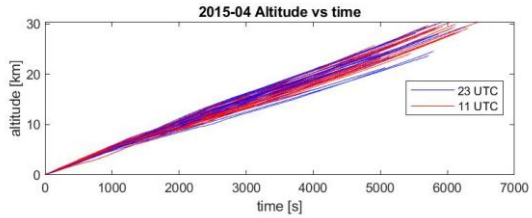


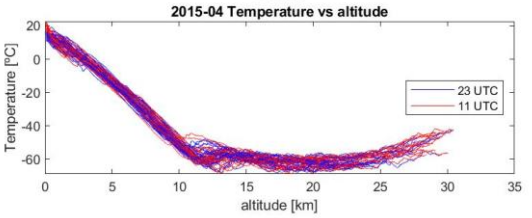


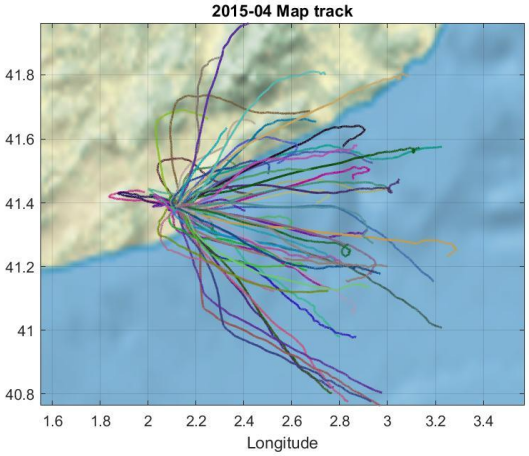


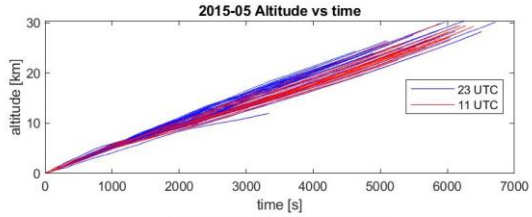


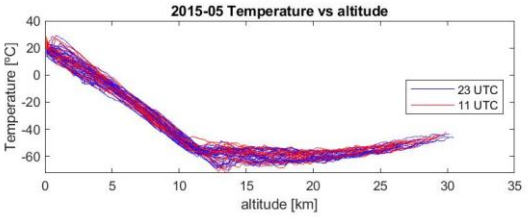


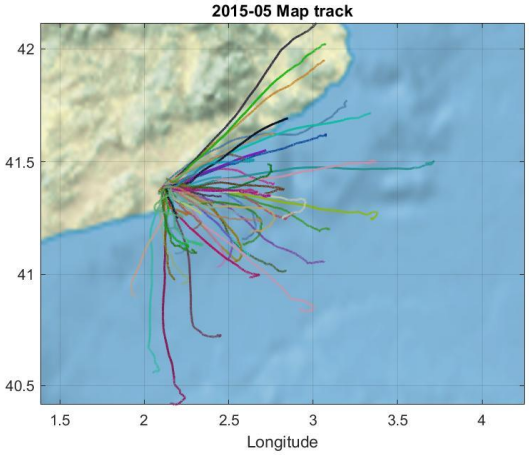

2015-04 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

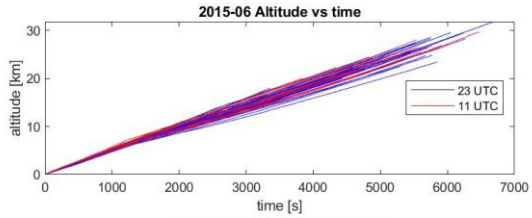

2015-04 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

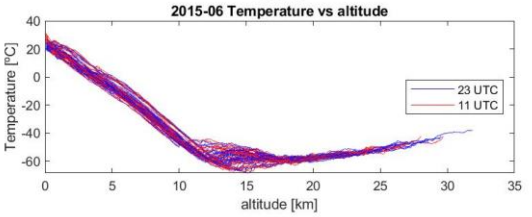

2015-04 Map track
Longitude

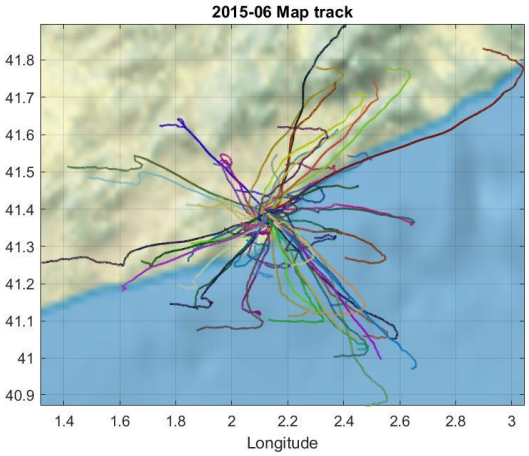

2015-05 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

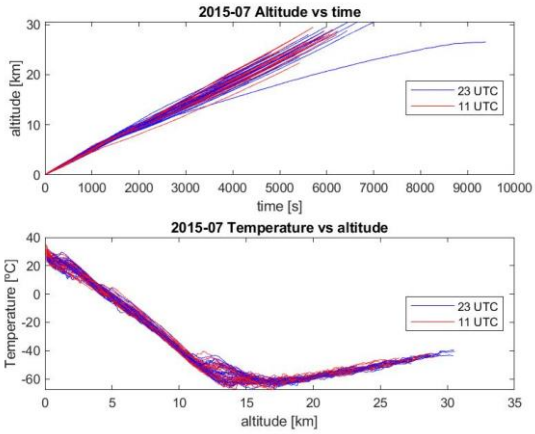

2015-05 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC


2015-05 Map track
Longitude

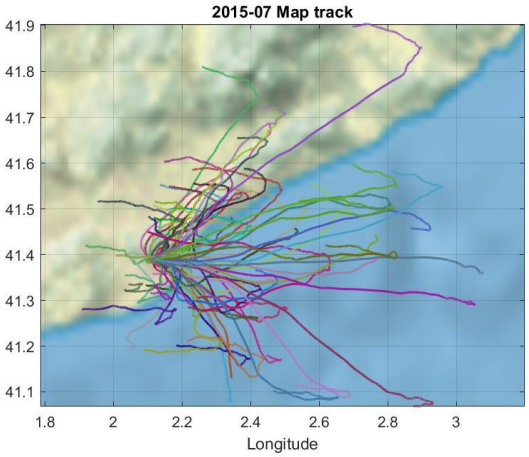

2015-06 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

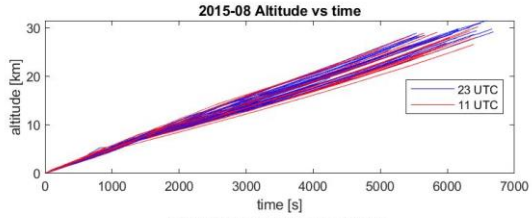

2015-06 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

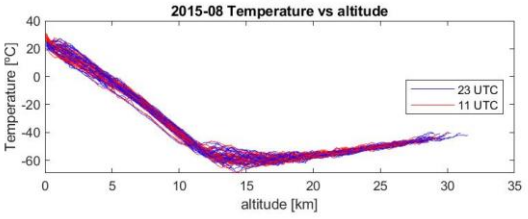

2015-06 Map track
Longitude

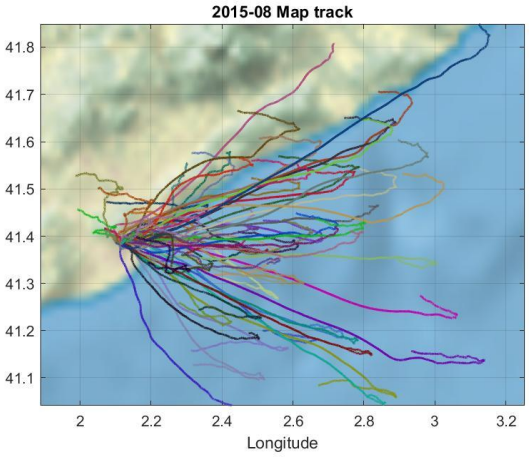

2015-07 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

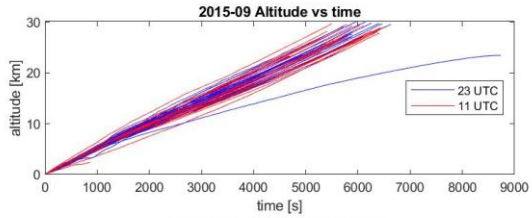
2015-07 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

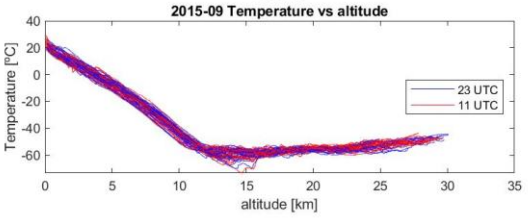

2015-07 Map track
Longitude

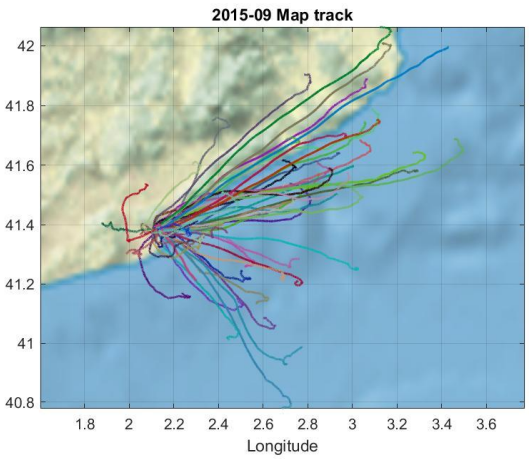

2015-08 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC


2015-08 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC


2015-08 Map track
Longitude


2015-09 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC


2015-09 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC


2015-09 Map track
Longitude

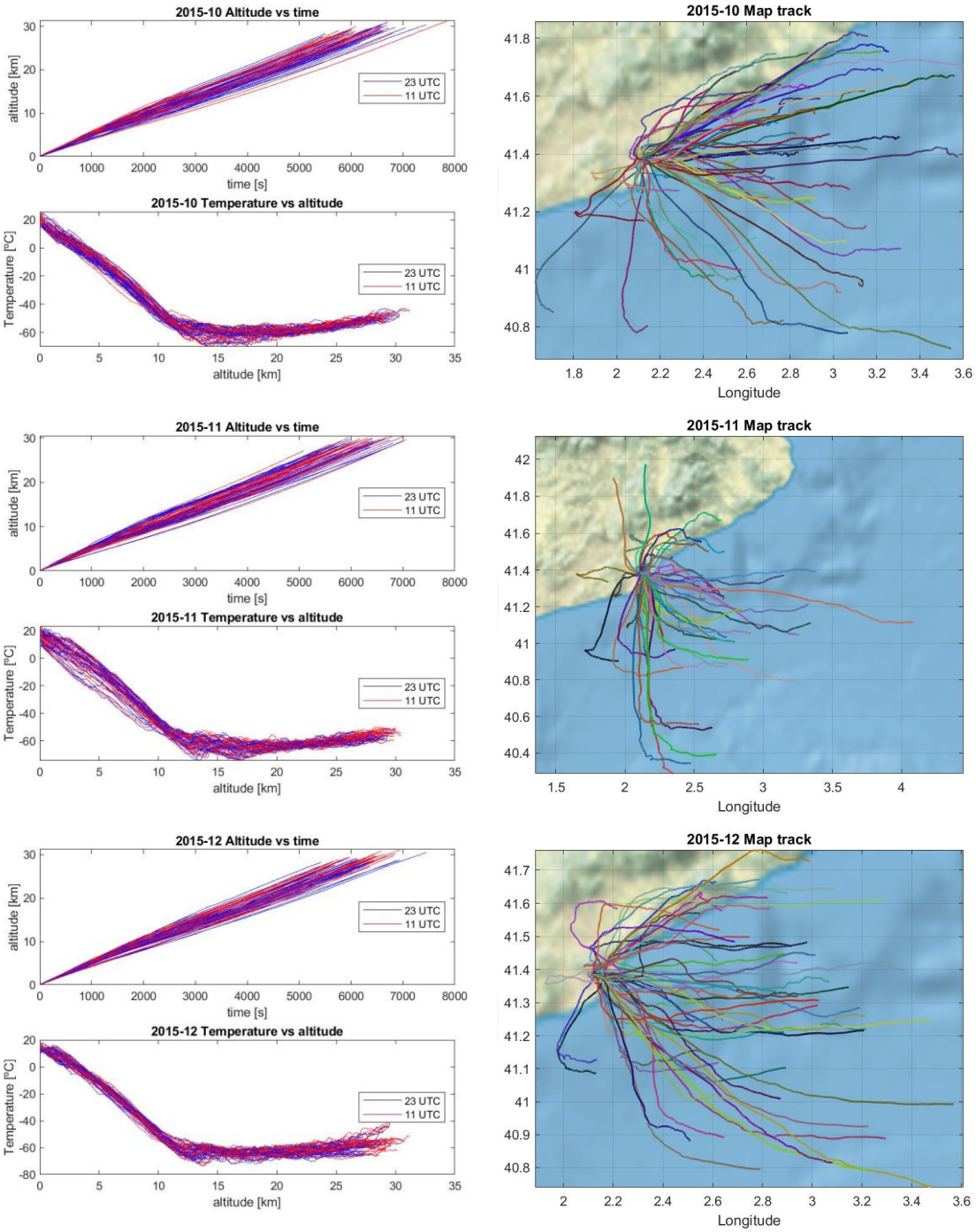
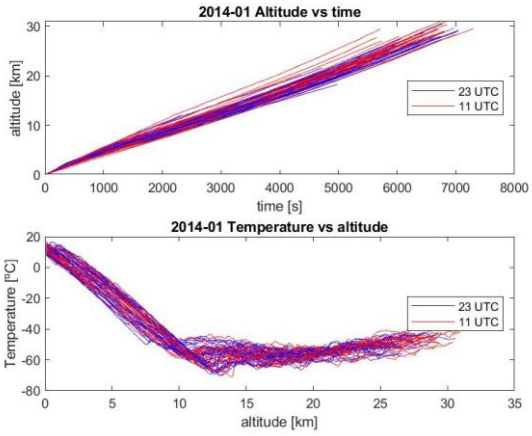
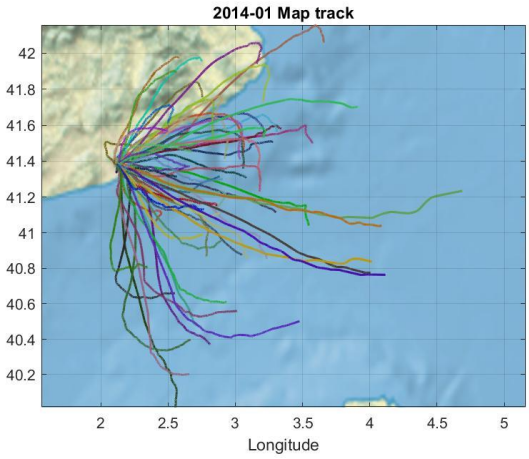
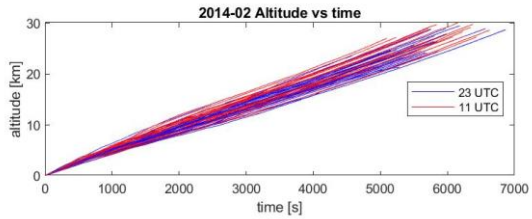


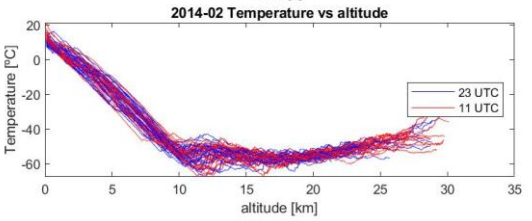
Figure 3 2015 Month by month altitude vs time and temperature vs altitude and flight trajectory

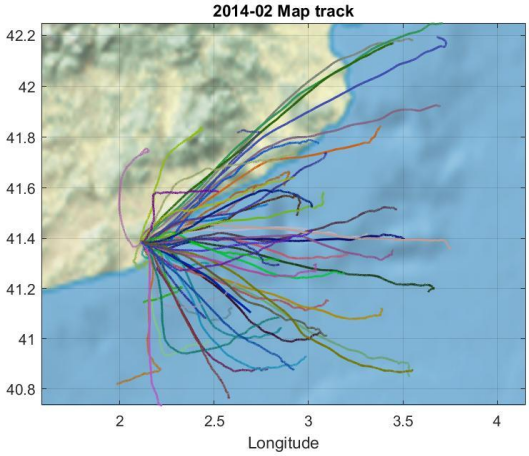
A.2.4. Year 2014 study

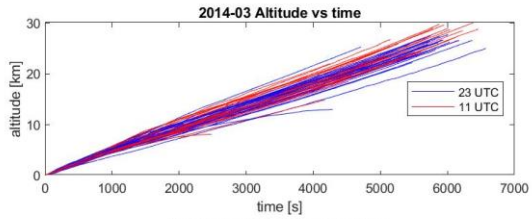


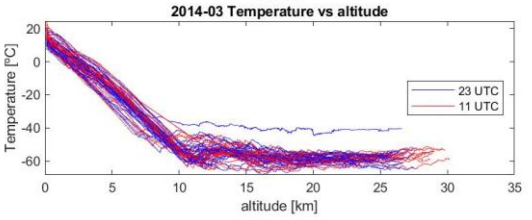


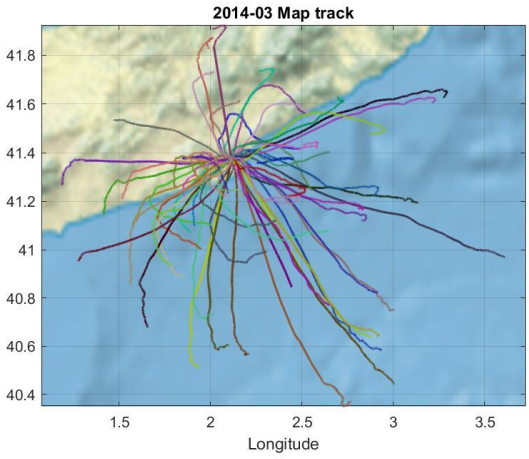


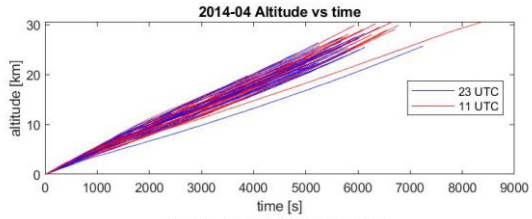


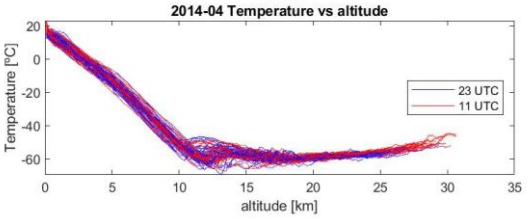


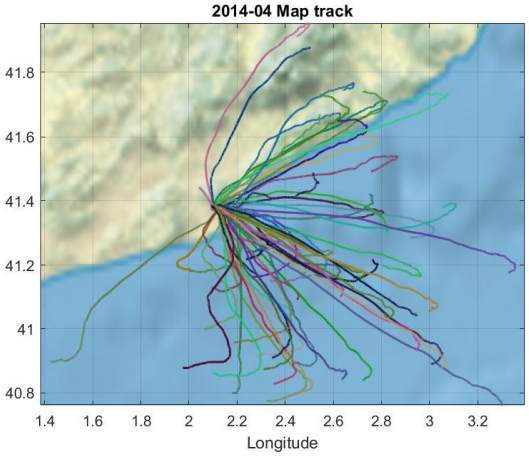


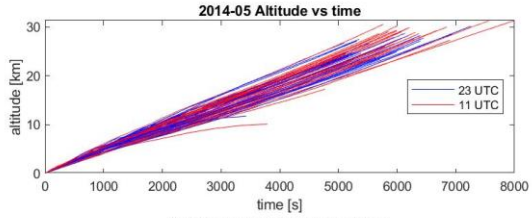


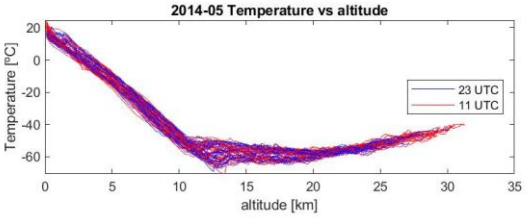


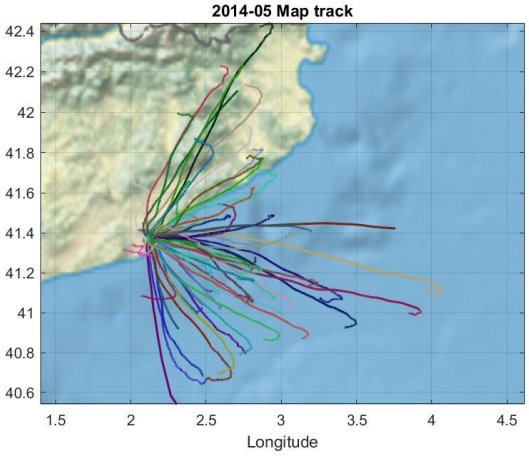

2014-04 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

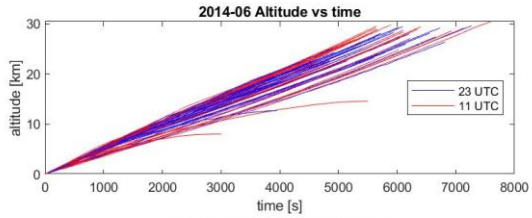

2014-04 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

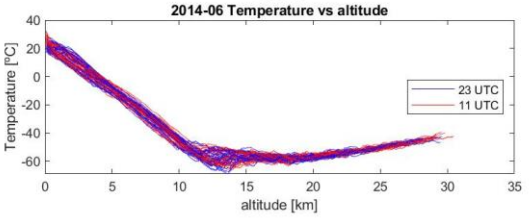

2014-04 Map track
Longitude

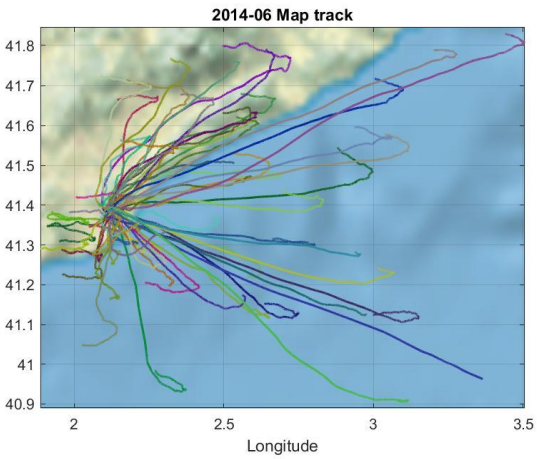

2014-05 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

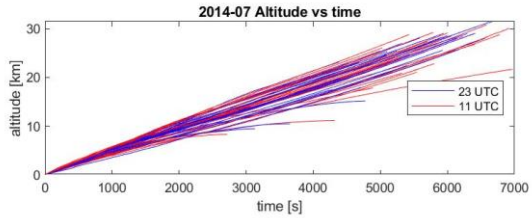

2014-05 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

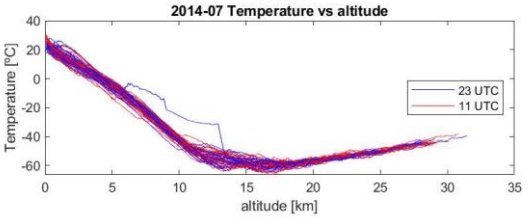

2014-05 Map track
Longitude

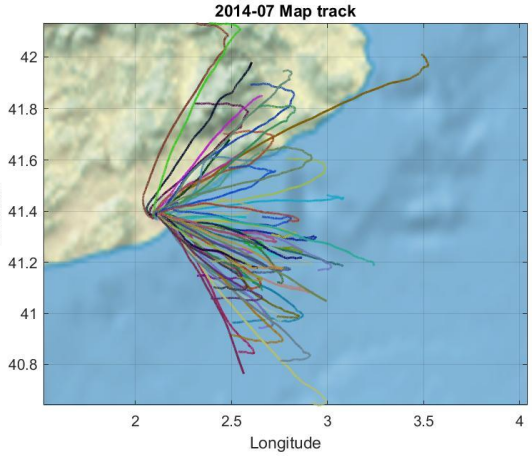

2014-06 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

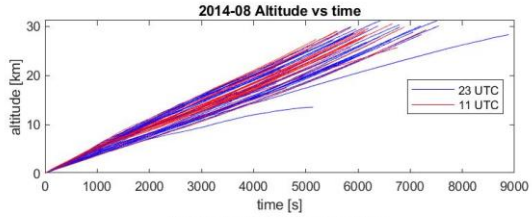

2014-06 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

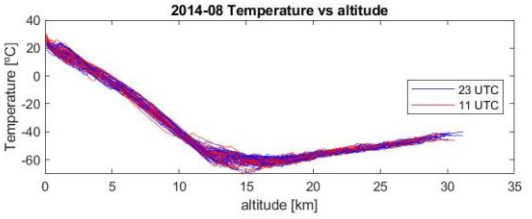

2014-06 Map track
Longitude

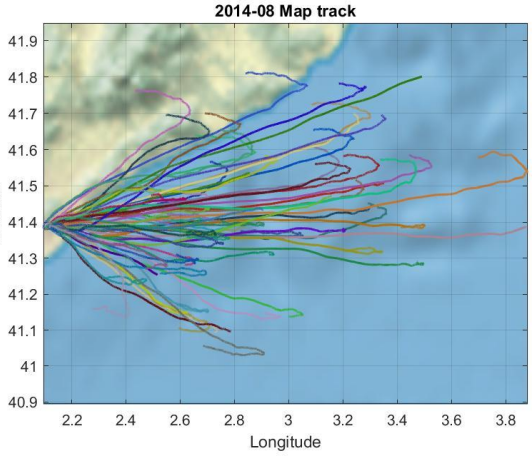

2014-07 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

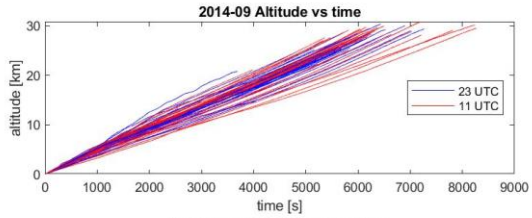

2014-07 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

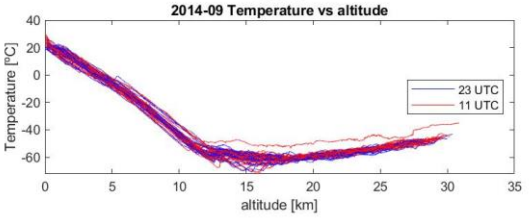

2014-07 Map track
Longitude

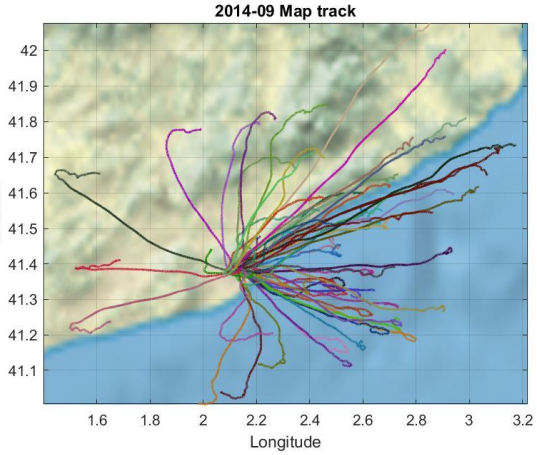

2014-08 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC


2014-08 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC


2014-08 Map track
Longitude


2014-09 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC


2014-09 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC


2014-09 Map track
Longitude

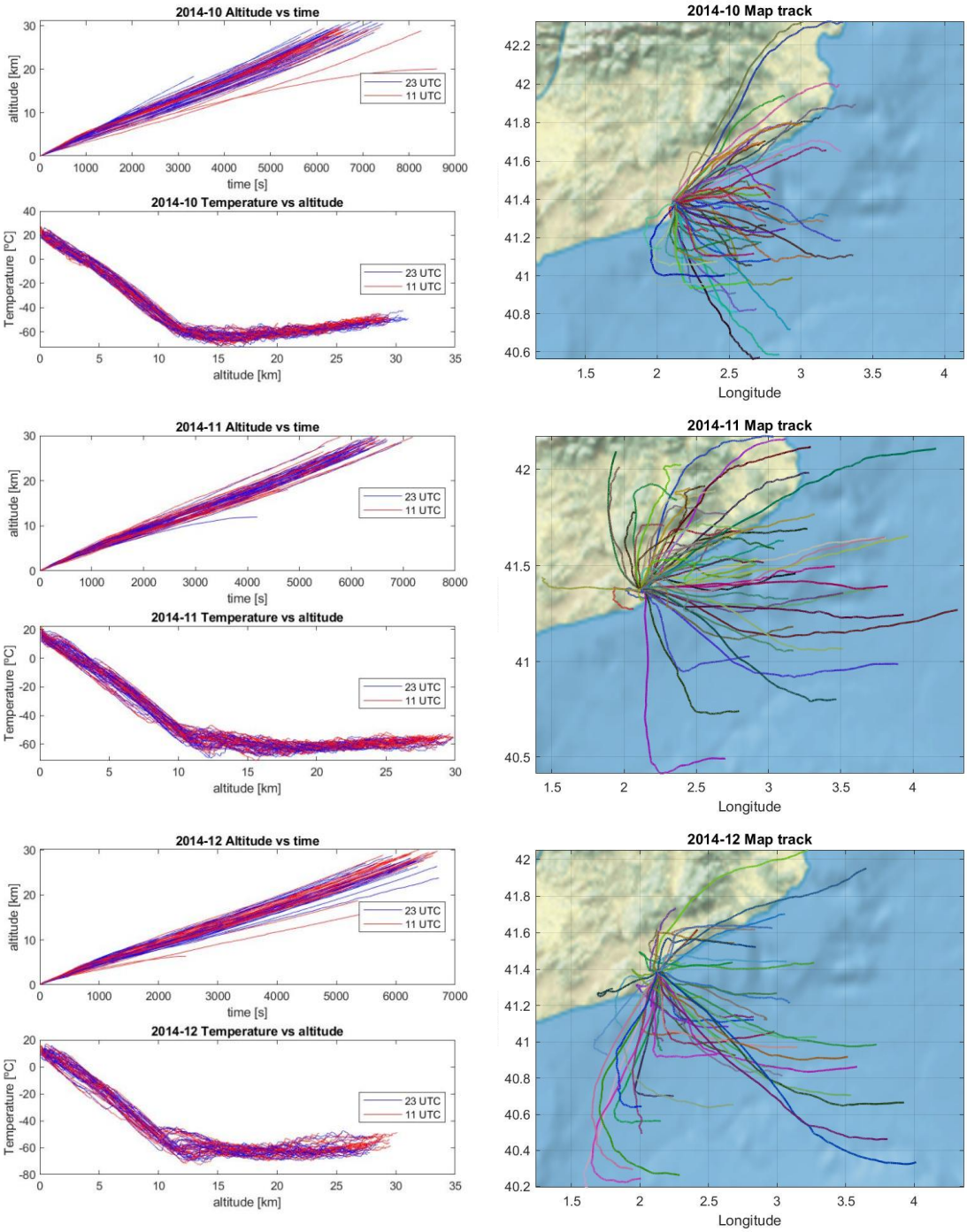
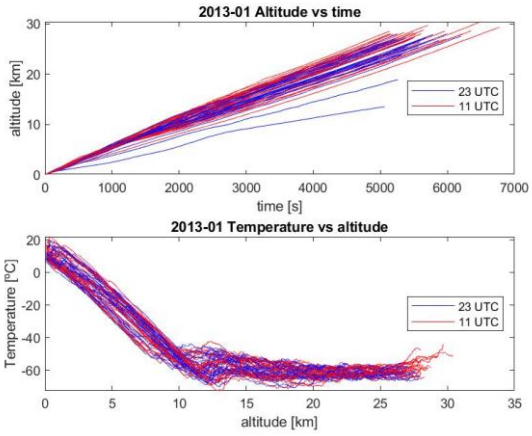
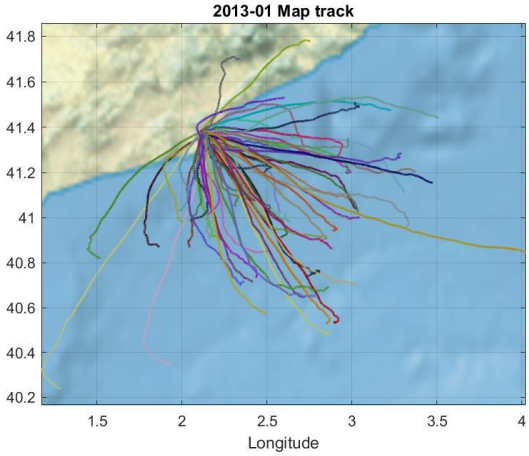
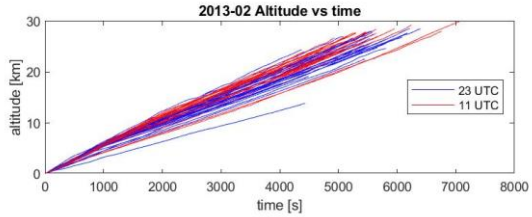


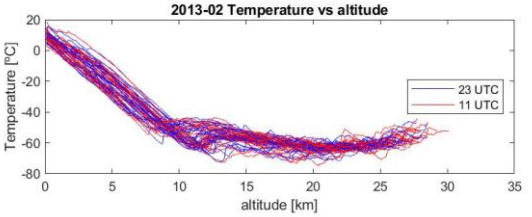
Figure 4 2014 Month by month altitude vs time and temperature vs altitude and flight trajectory

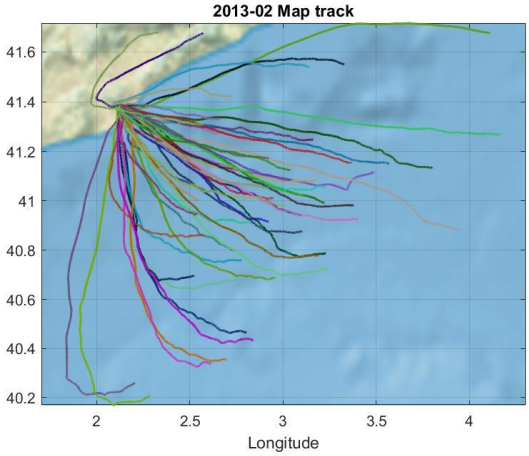
A.2.5. Year 2013 study

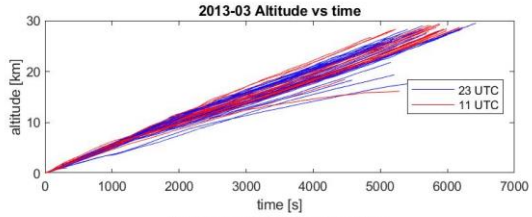


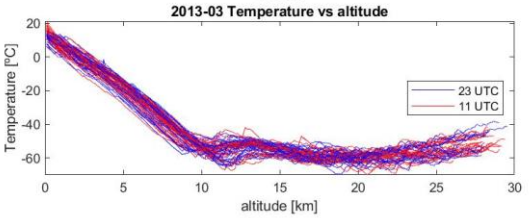


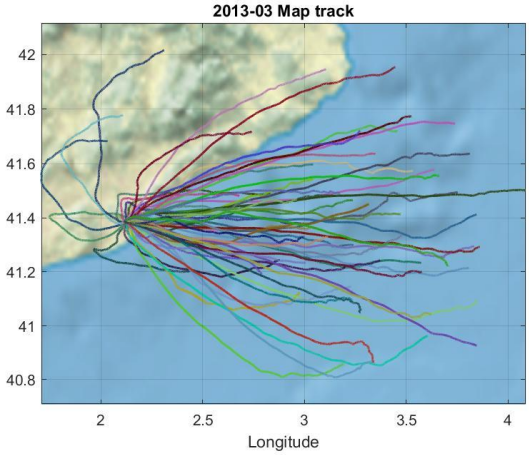


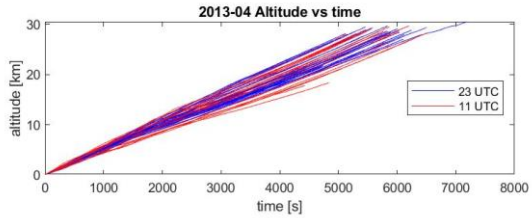


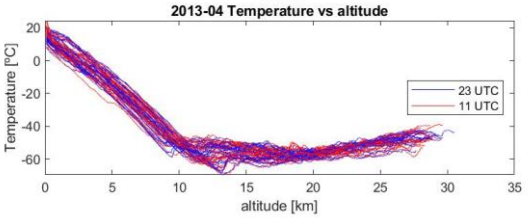


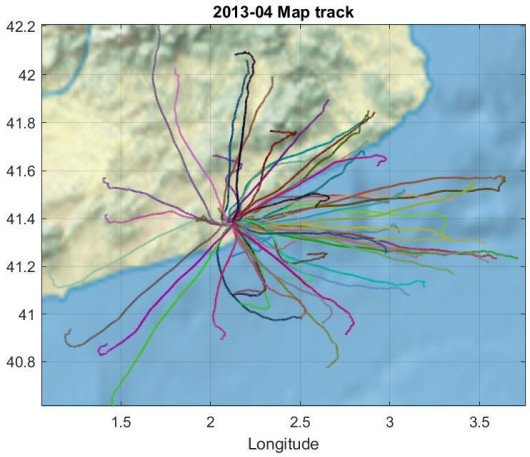


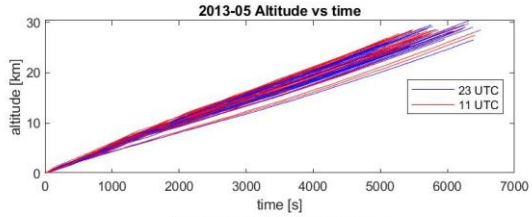


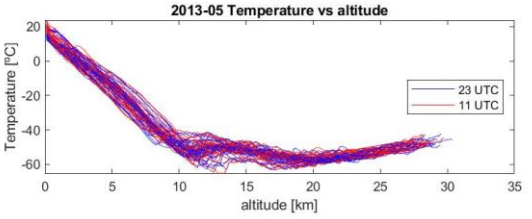


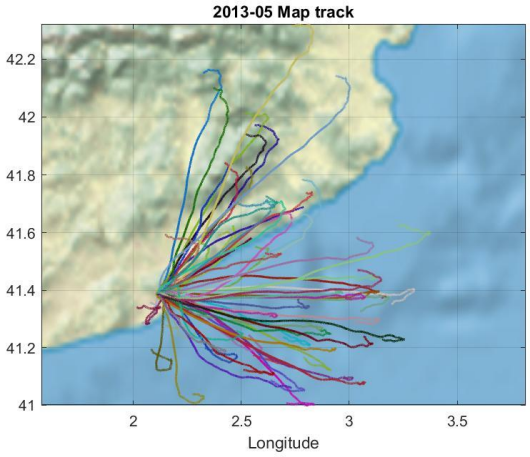

2013-04 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

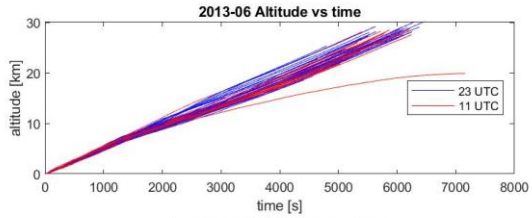

2013-04 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

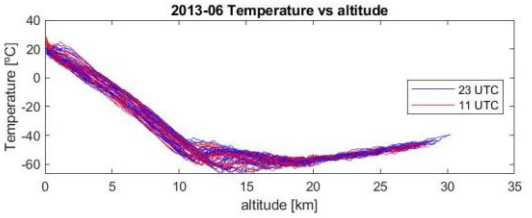

2013-04 Map track
Longitude

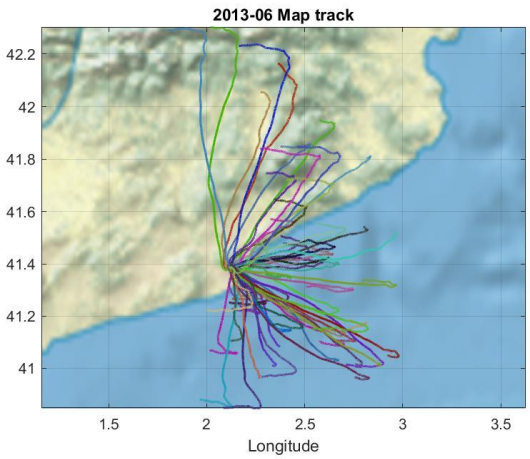

2013-05 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

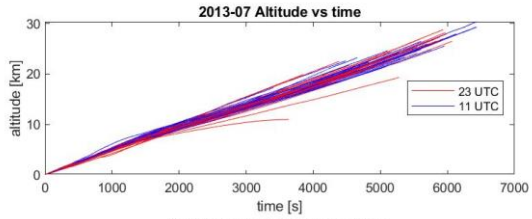

2013-05 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

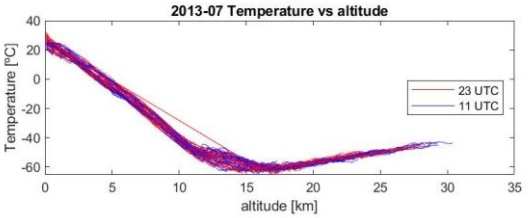

2013-05 Map track
Longitude

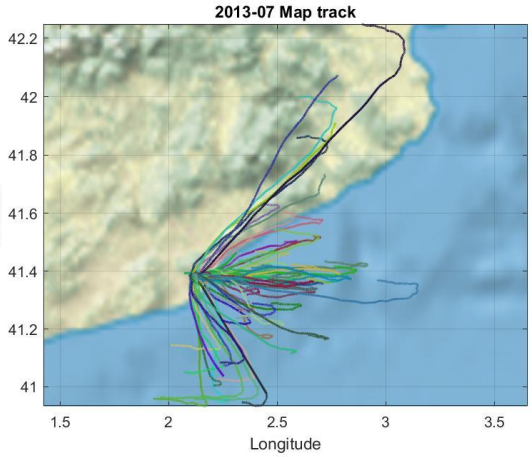

2013-06 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

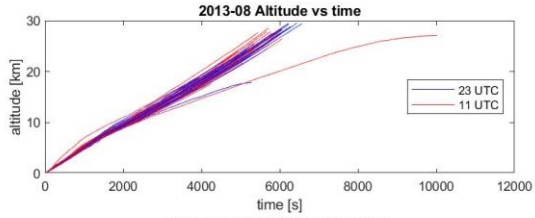

2013-06 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

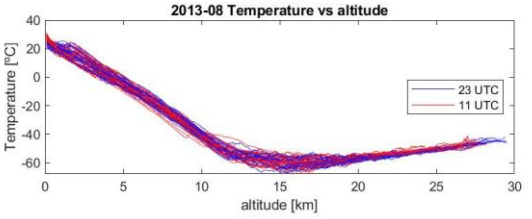

2013-06 Map track
Longitude

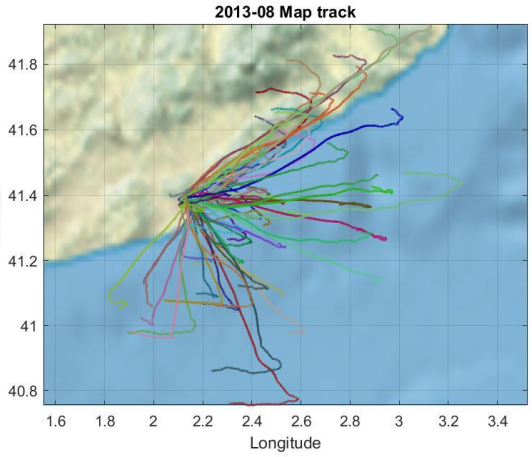

2013-07 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC

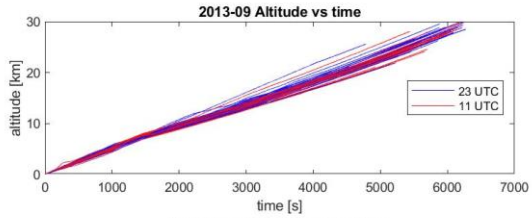

2013-07 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC

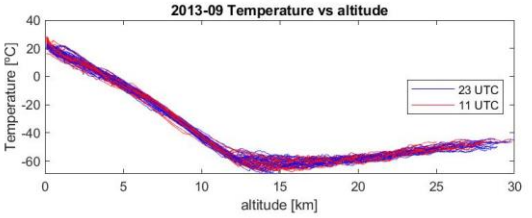

2013-07 Map track
Longitude

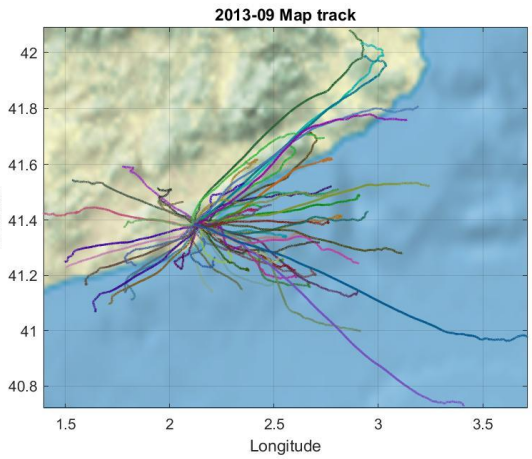

2013-08 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC


2013-08 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC


2013-08 Map track
Longitude


2013-09 Altitude vs time
altitude [km]
time [s]
23 UTC
11 UTC


2013-09 Temperature vs altitude
Temperature [°C]
altitude [km]
23 UTC
11 UTC


2013-09 Map track
Longitude

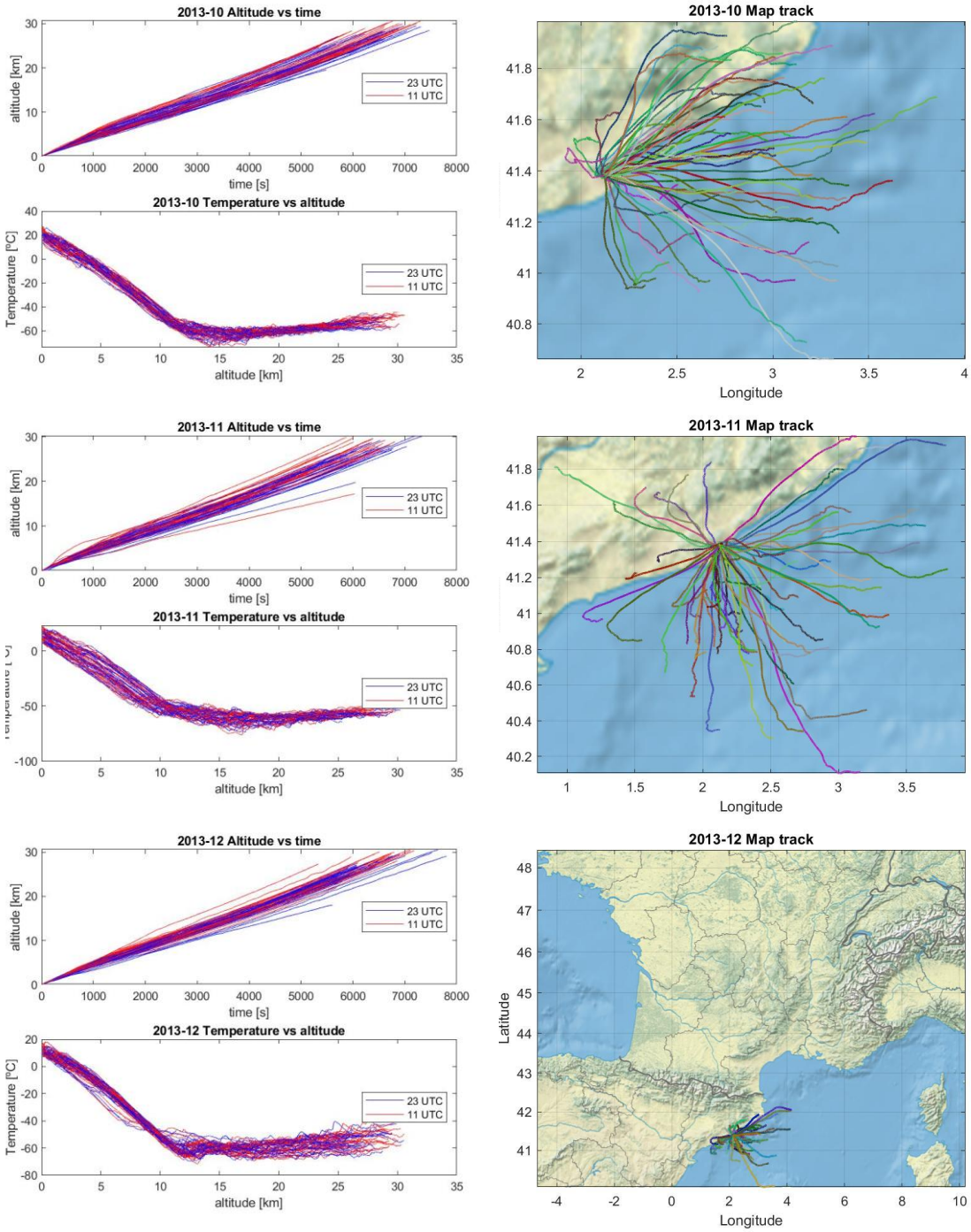


Figure 5 2013 Month by month altitude vs time and temperature vs altitude and flight trajectory

If we look at the ascension profiles, it is seen that there's a variation in the slope in both curves, at around 11 km of altitude. This is expected, at it is the altitude of the tropopause, the boundary between the troposphere and the stratosphere layers. It is characterized by a temperature inversion (a layer of warm air above a colder one) in most places of the globe, while on others it forms a zone isothermal with altitude. It is clearly seen in both 1976 Standard Atmosphere and NRMLSISE atmosphere models, seen in Figure 6:

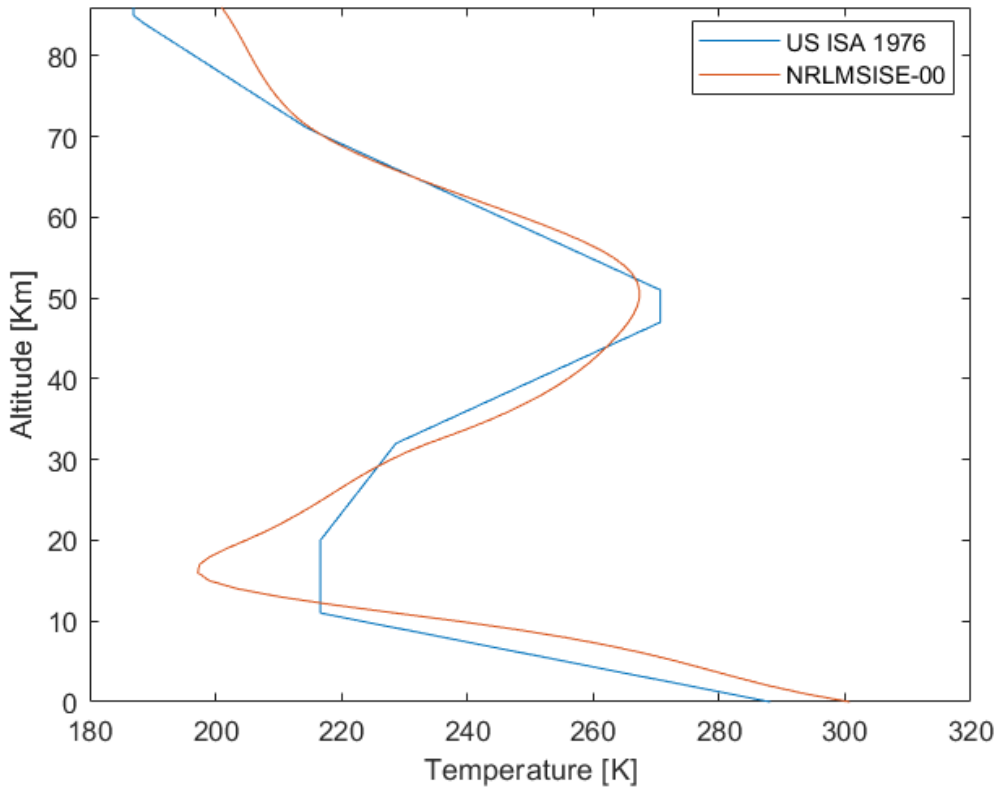


Figure 6 1976 Standard atmosphere and NRMLSISE atmosphere models

A.2.6. Statistical study

In order to determine if a mean for each month would be representative, a statistical study is performed month by month for each yea. To do so, a population of data has been studied, in increments of 15 minutes for each launch in every month. Then, a boxplot is plotted for each population, in order to see the dispersion. In the following pages these boxplots are plotted for each year and month.

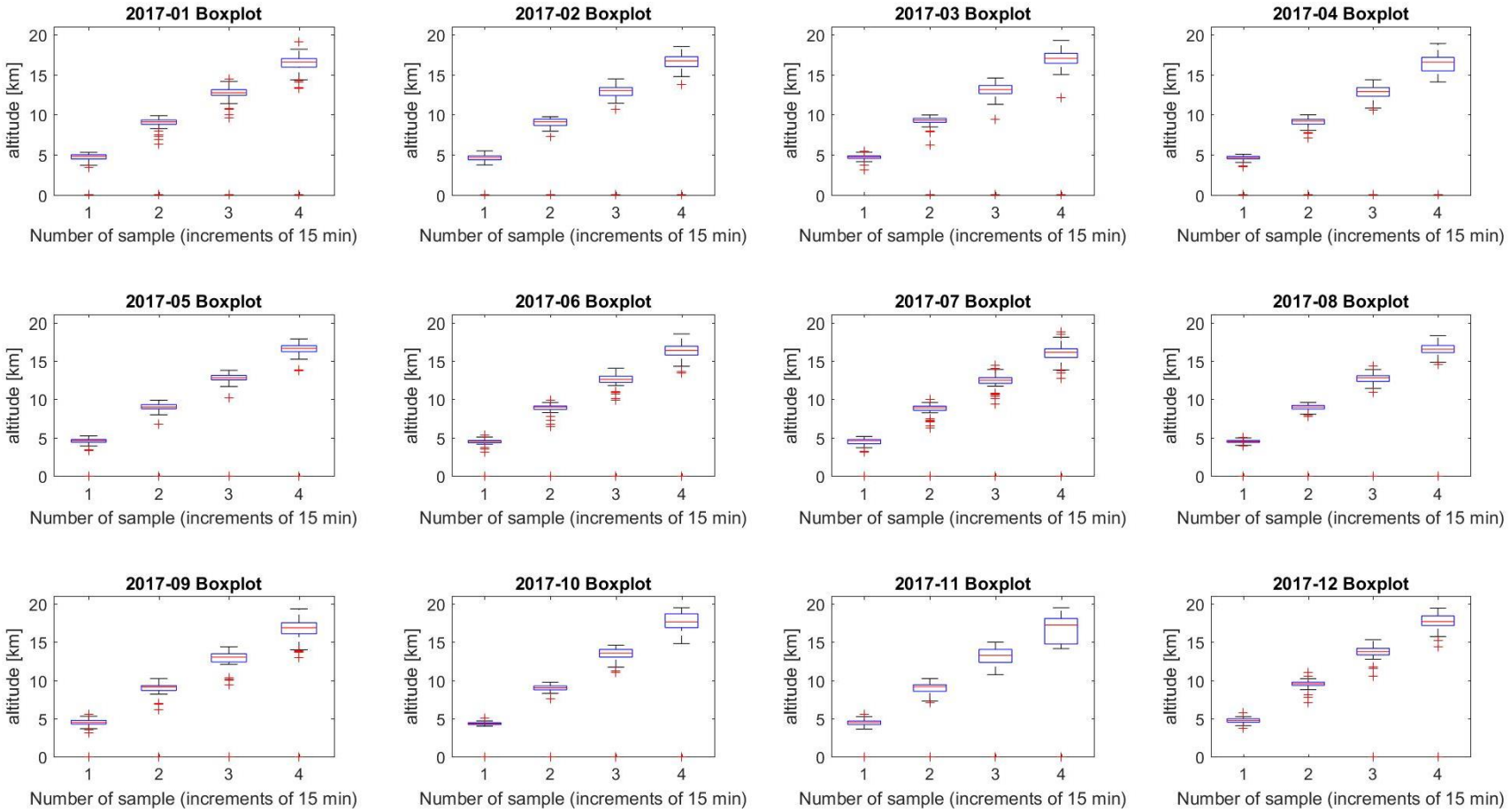


Figure 7 Year 2017 month by month statistical boxplots

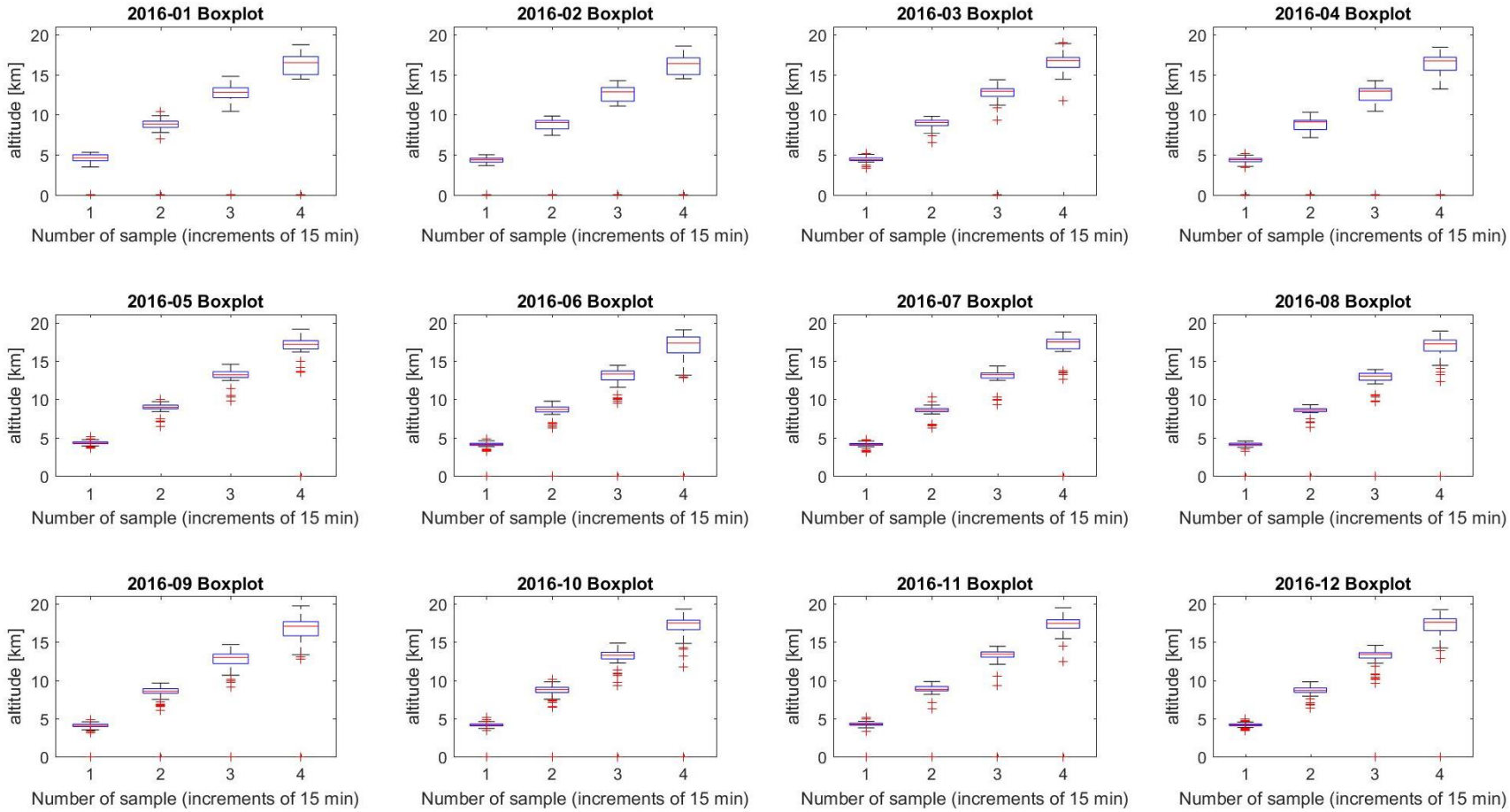


Figure 8 Year 2016 month by month statistical boxplots

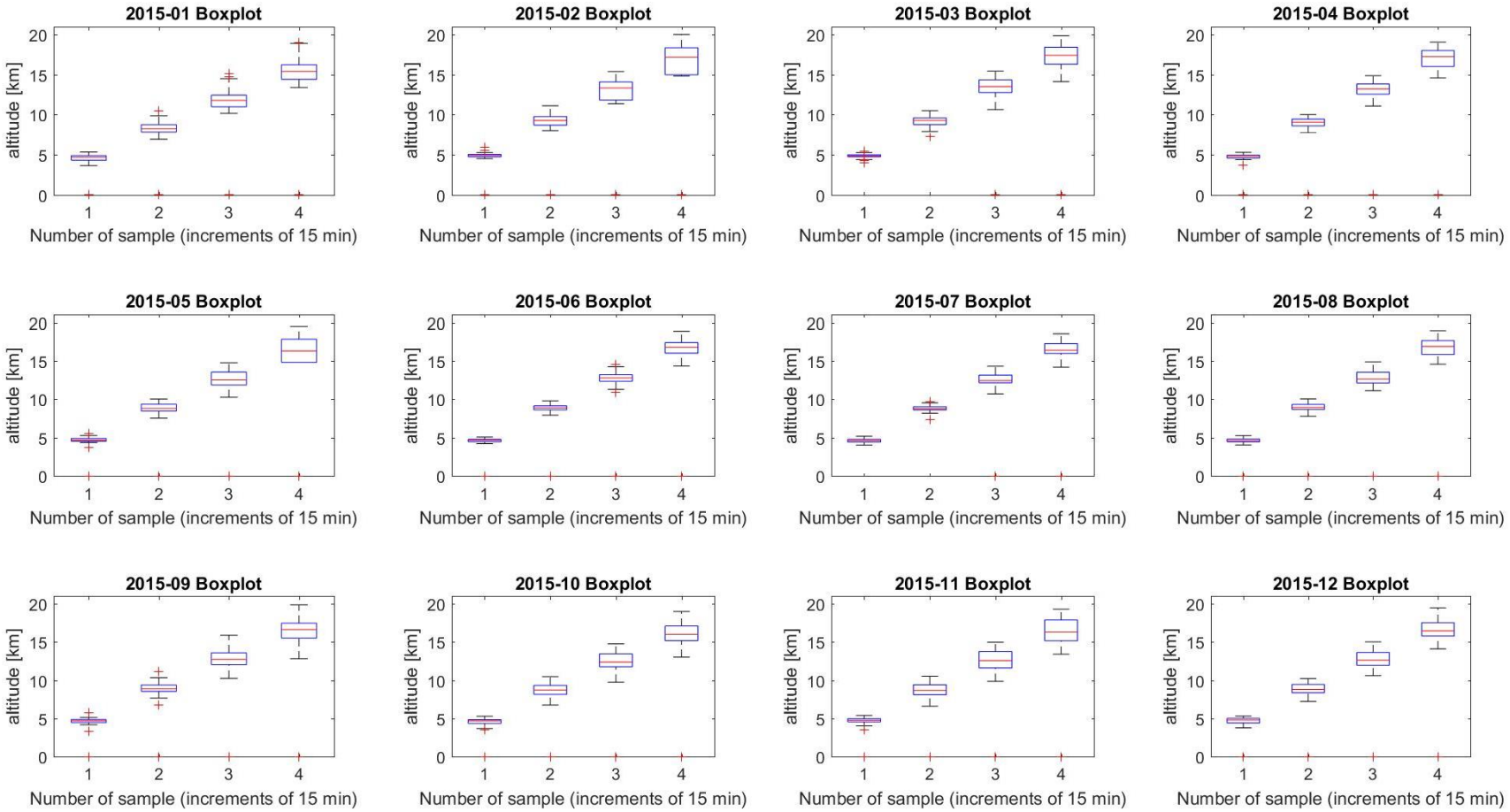


Figure 9 Year 2015 month by month statistical boxplots

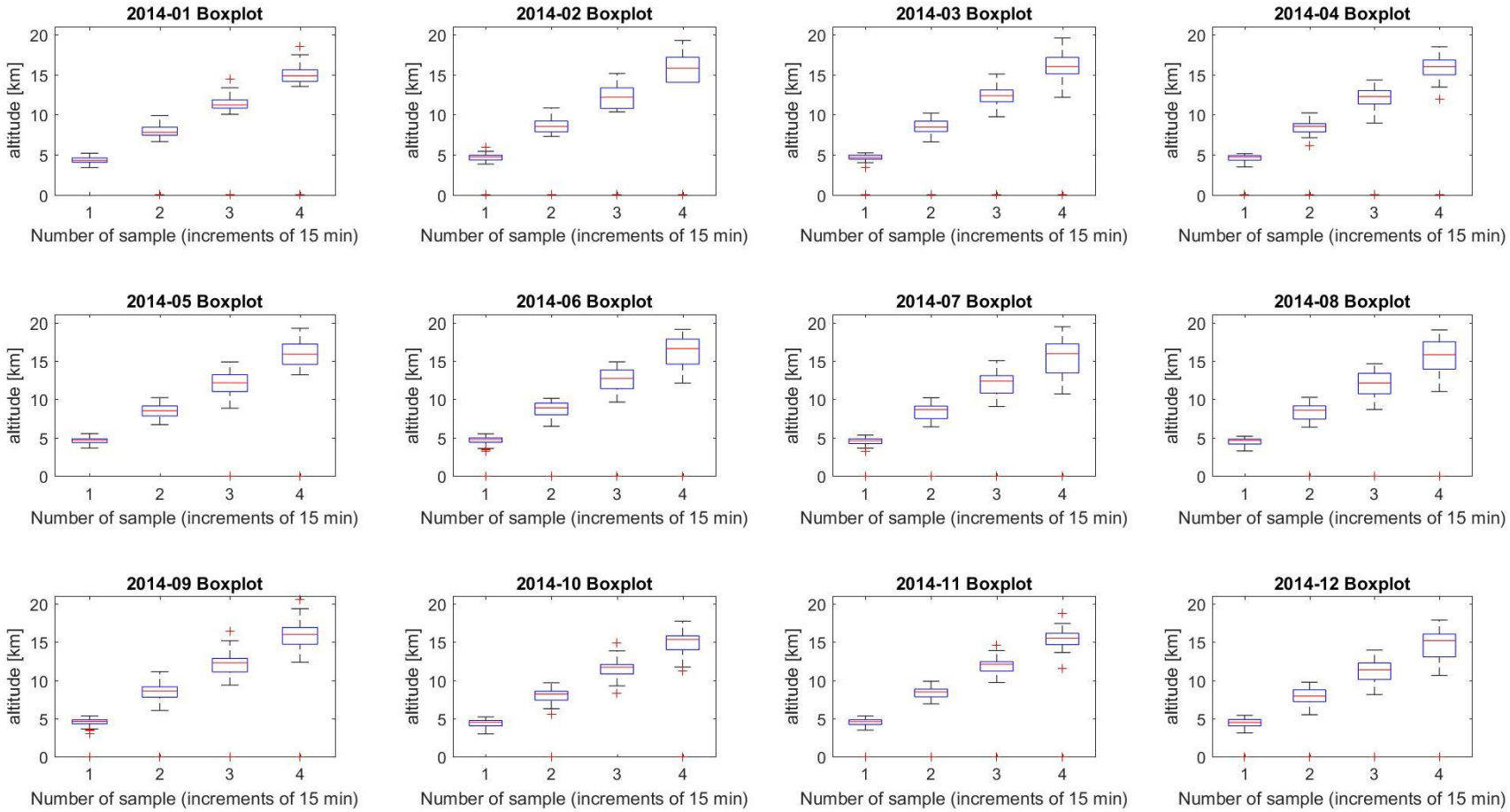


Figure 10 Year 2014 month by month statistical boxplots

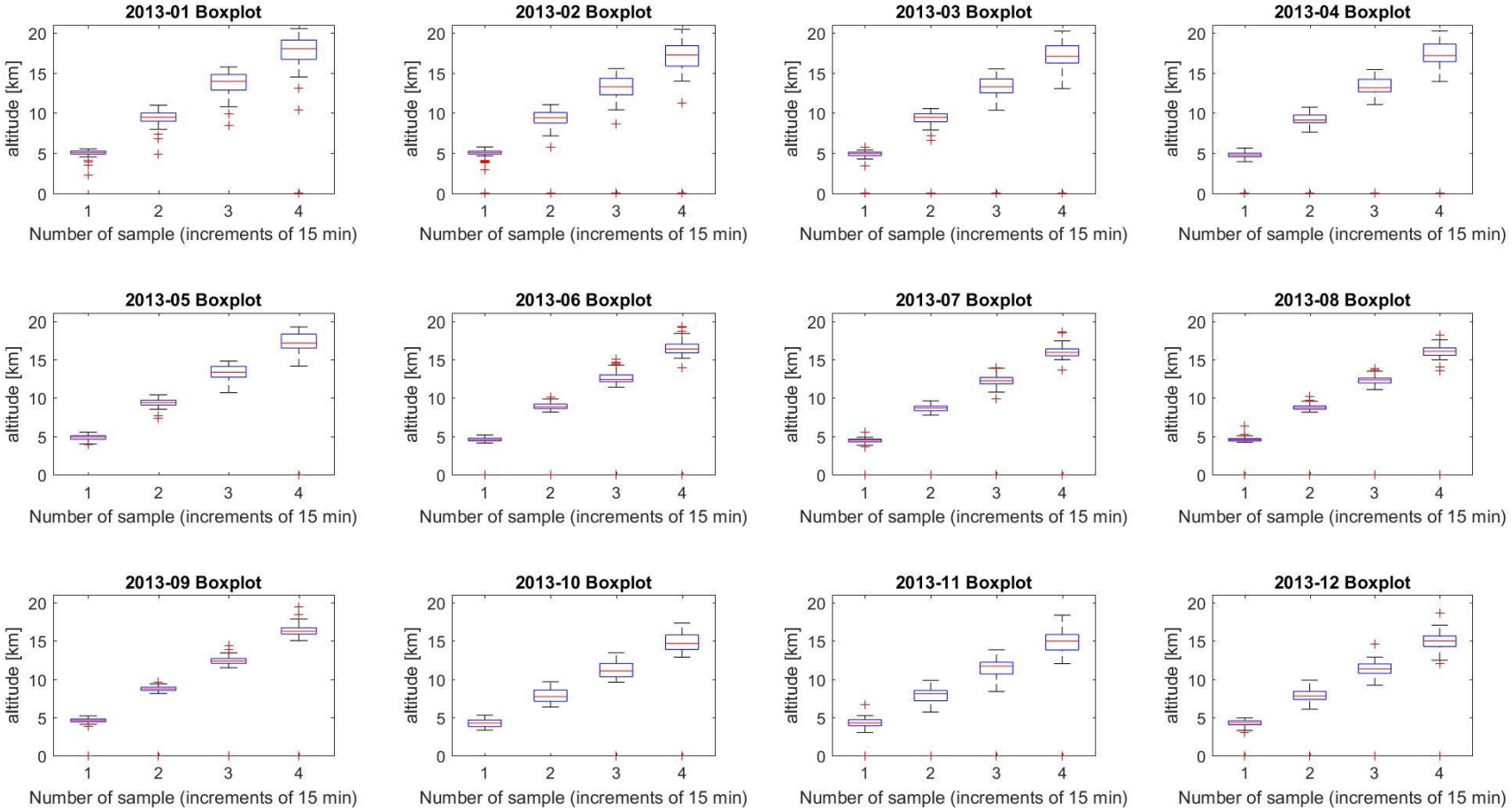


Figure 11 Year 2013 month by month statistical boxplots

It is seen that the dispersion increases as the balloon ascends, as expected (due to the different atmospheric conditions in each flight) although it is low enough to consider that the mean ascent profile for each month would be representative of the monthly launches. To be sure, the Pearson variation coefficient is calculated for each year-month and instant. Then, the maximum for each month is found and represented:

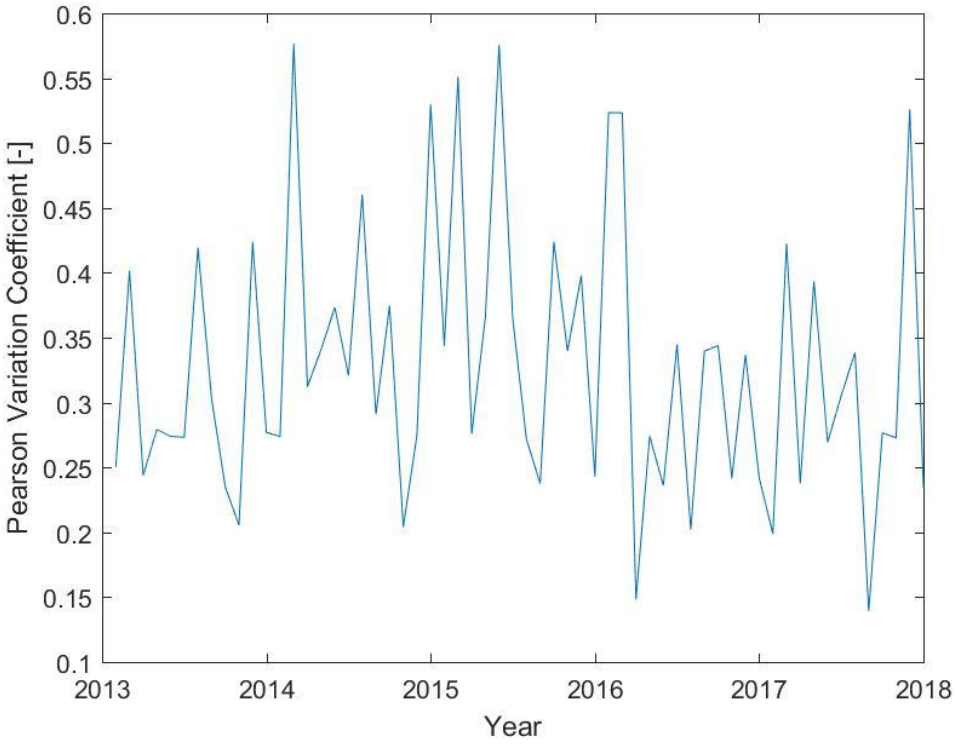


Figure 12 Pearson coefficient for studied years 2012-2017

As it is seen, the maximum is set at 57% while most oscile between 20% and 40%. This means the dispersions are low enough to make the means representative.

A.3. Final simulations fit

In this chapter, the pool of simulations for every studied launch is plotted and next the final simulation fit for each case.

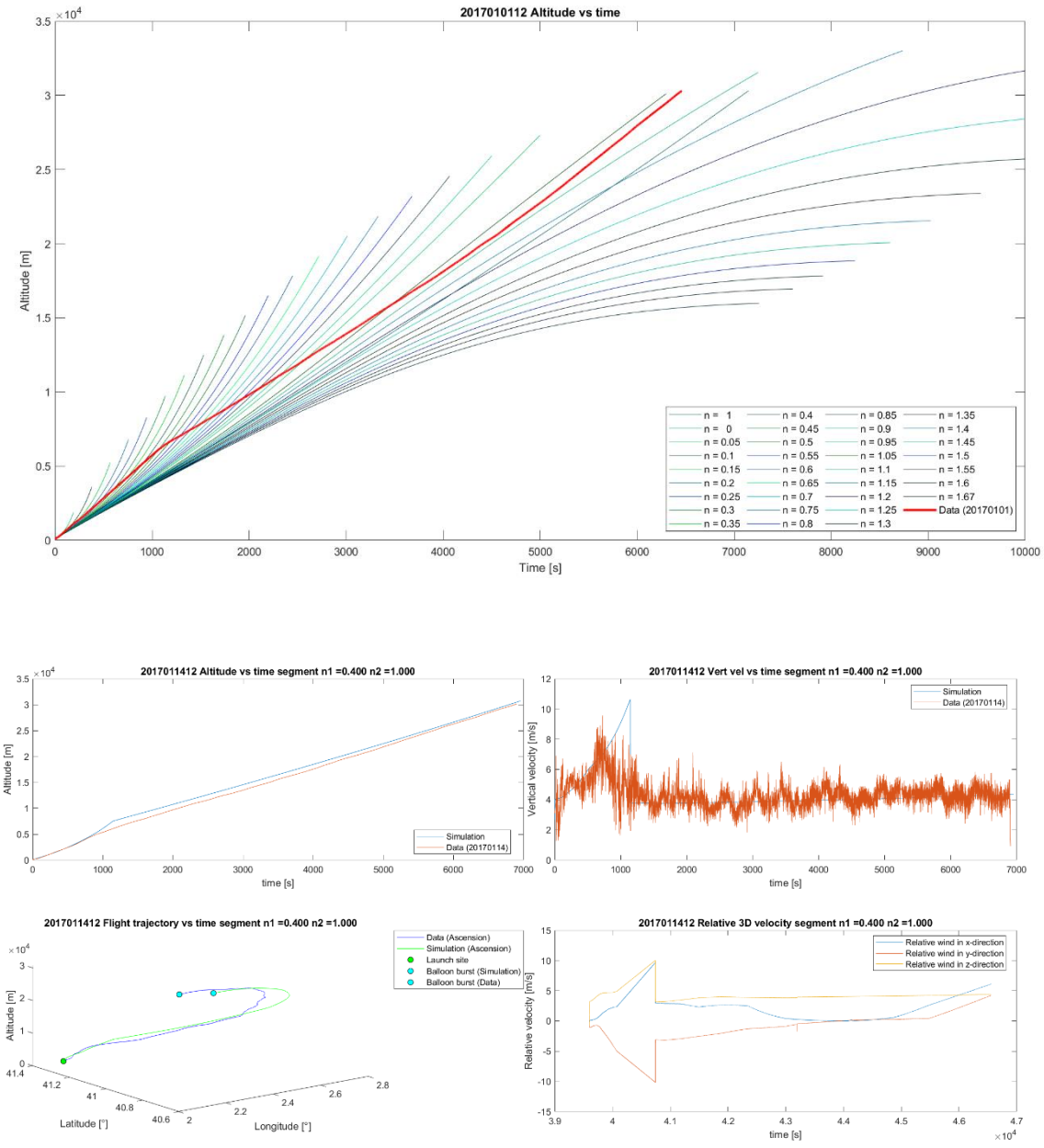


Figure 13 Fit for launch at 01/01/2017 11 UTC

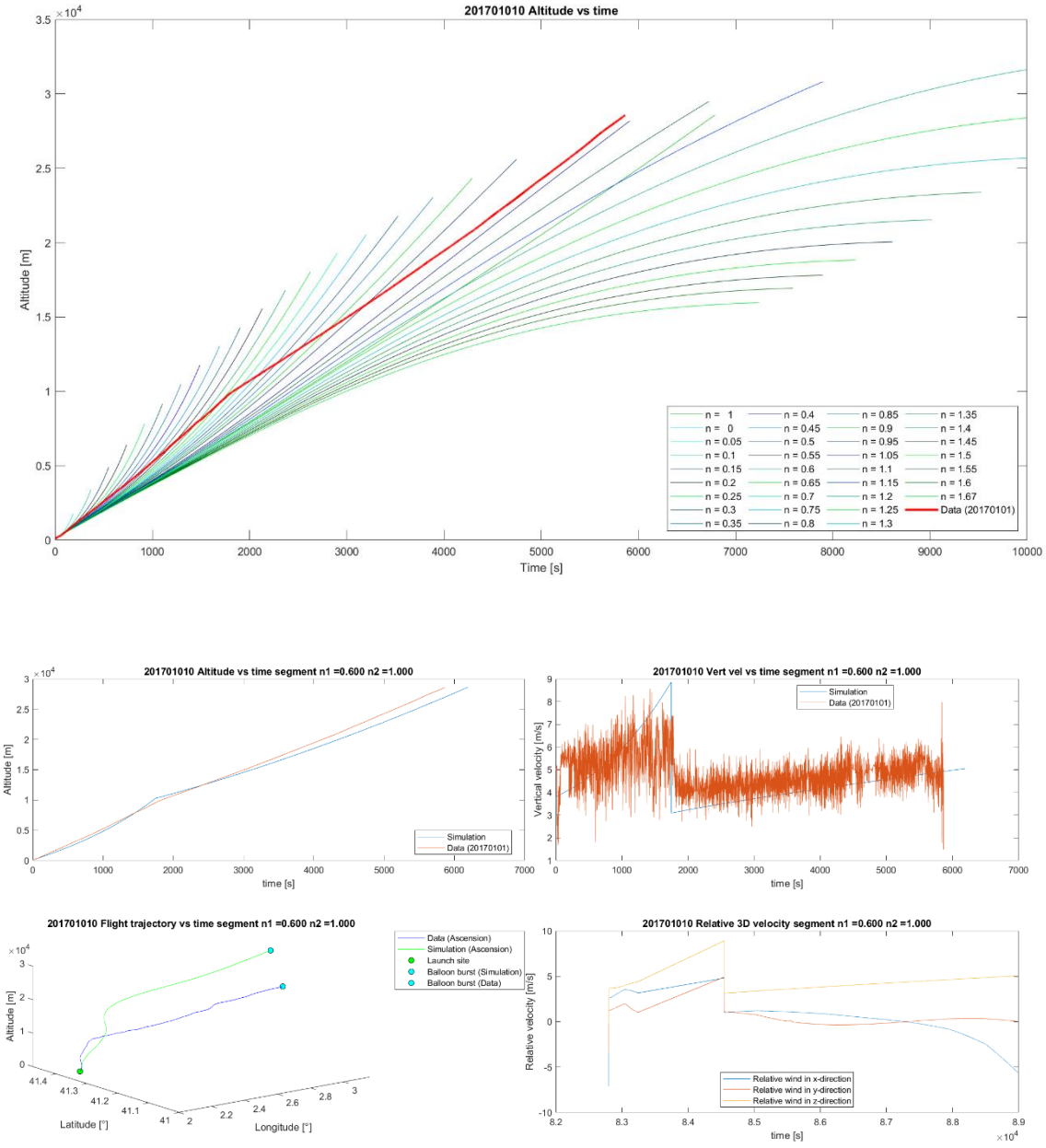


Figure 14 Fit for launch at 01/01/2017 23 UTC

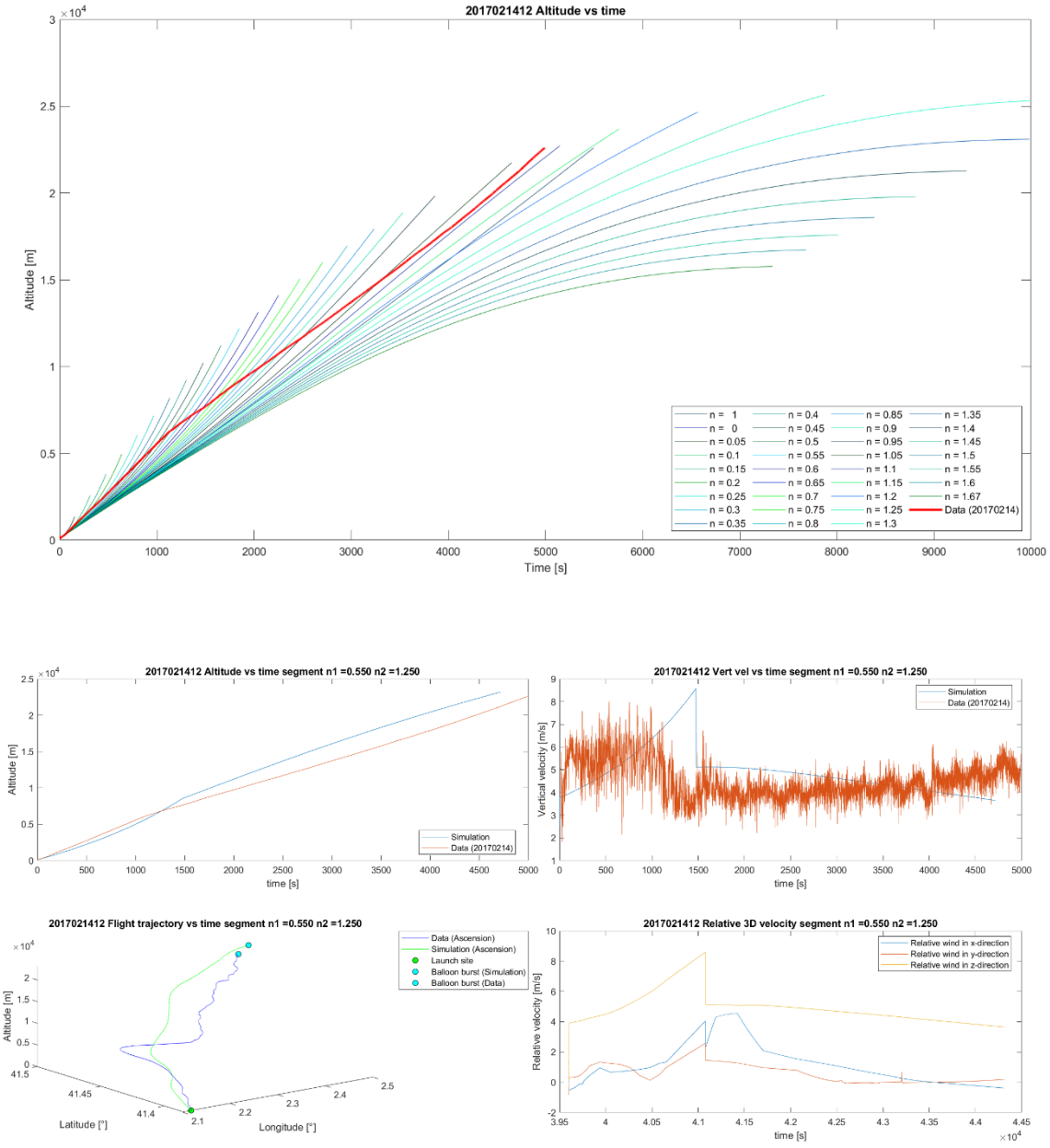


Figure 15 Fit for launch at 14/02/2017 11 UTC

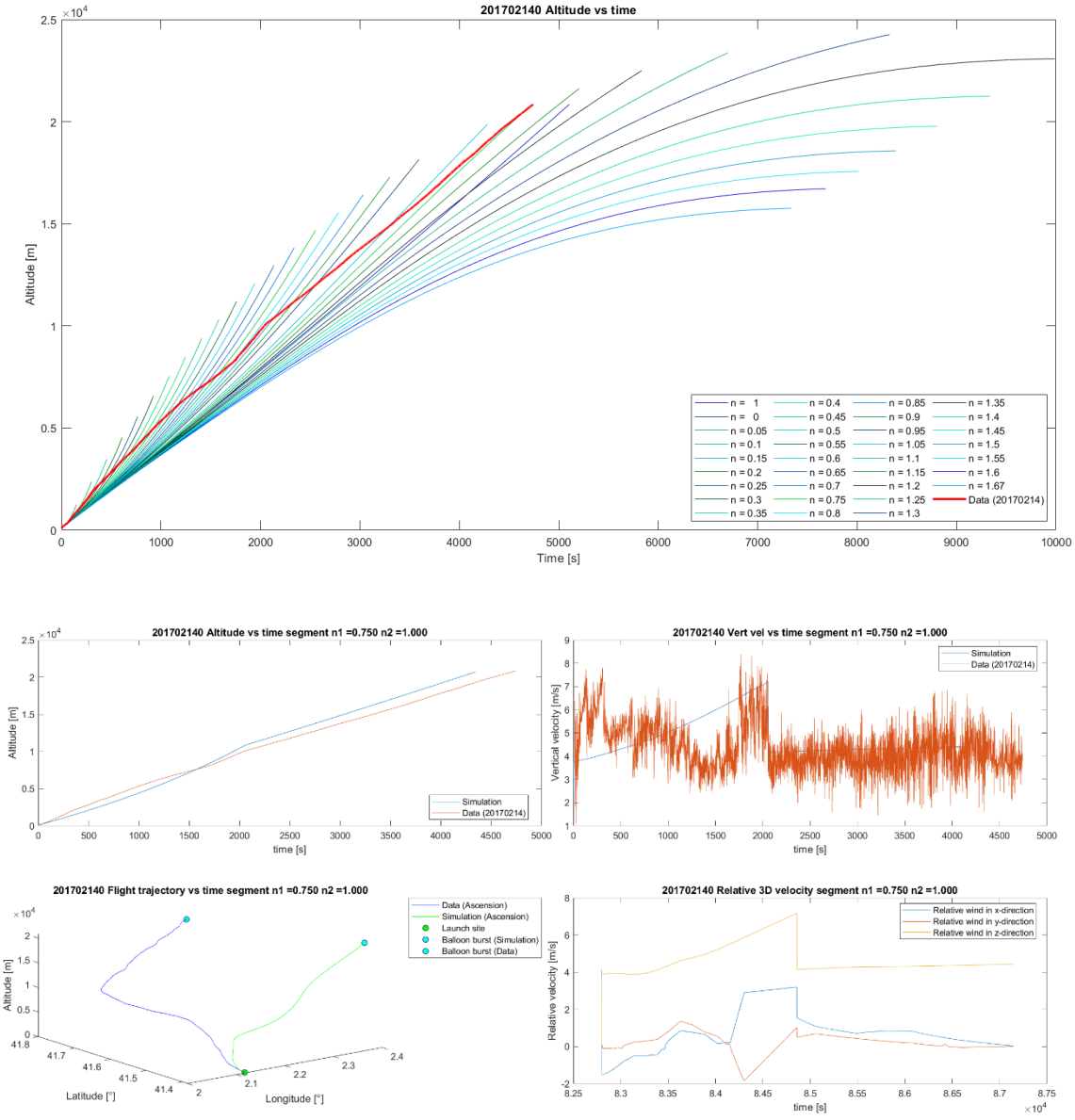


Figure 16 Fit for launch at 14/02/2017 23 UTC

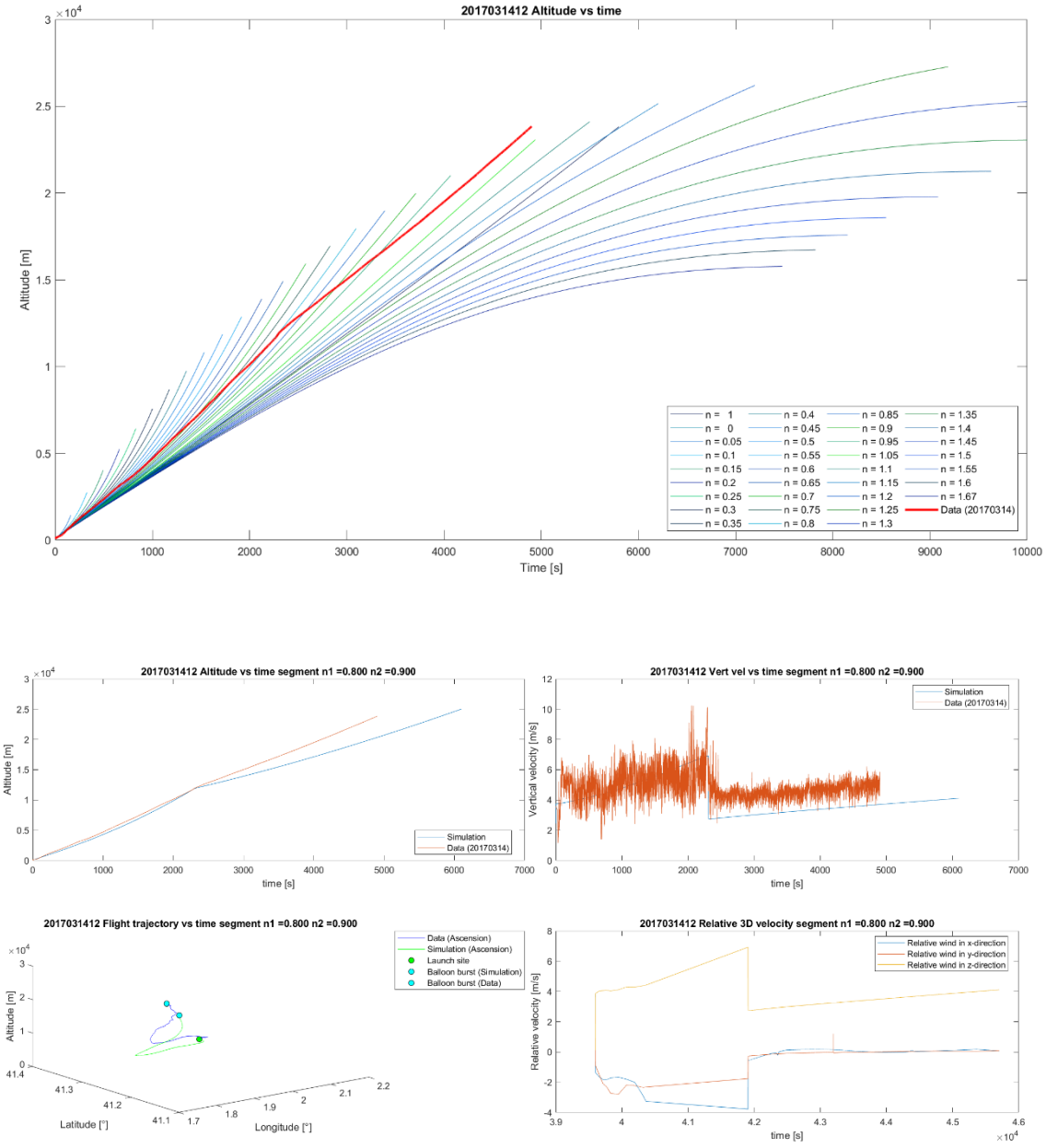


Figure 17 Fit for launch at 14/03/2017 11 UTC

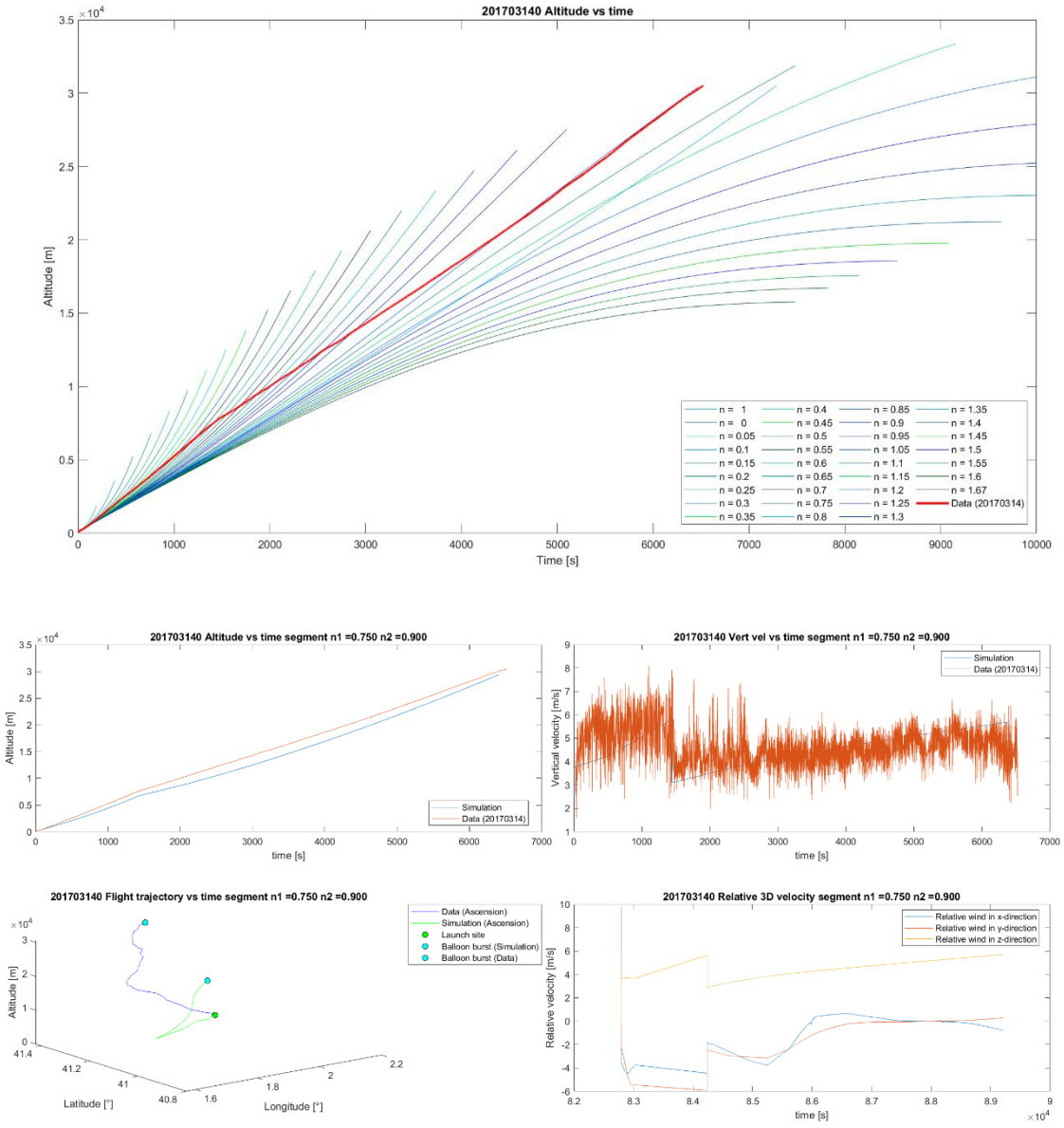


Figure 18 Fit for launch at 14/03/2017 23 UTC

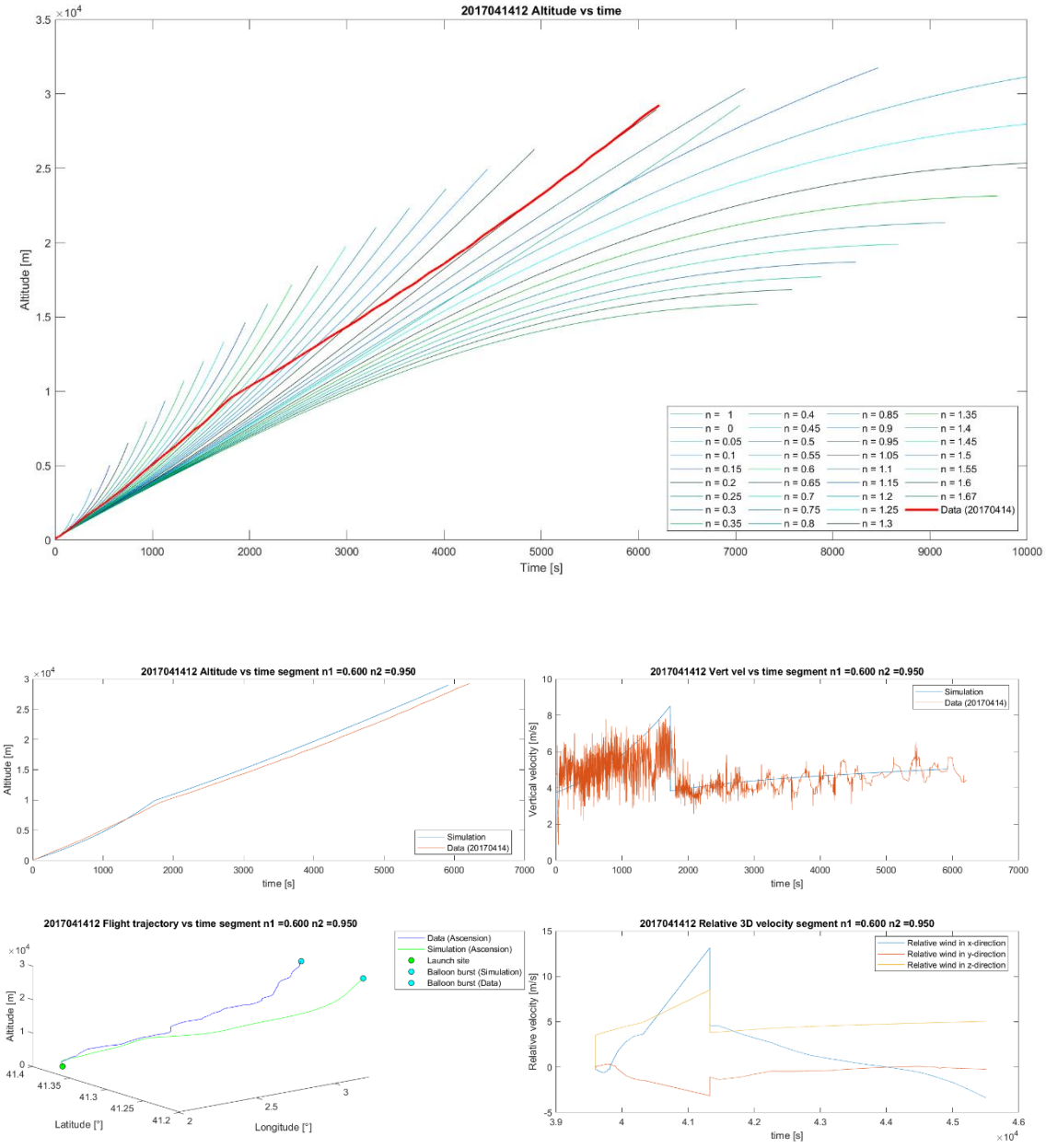


Figure 19 Fit for launch at 14/04/2017 11 UTC

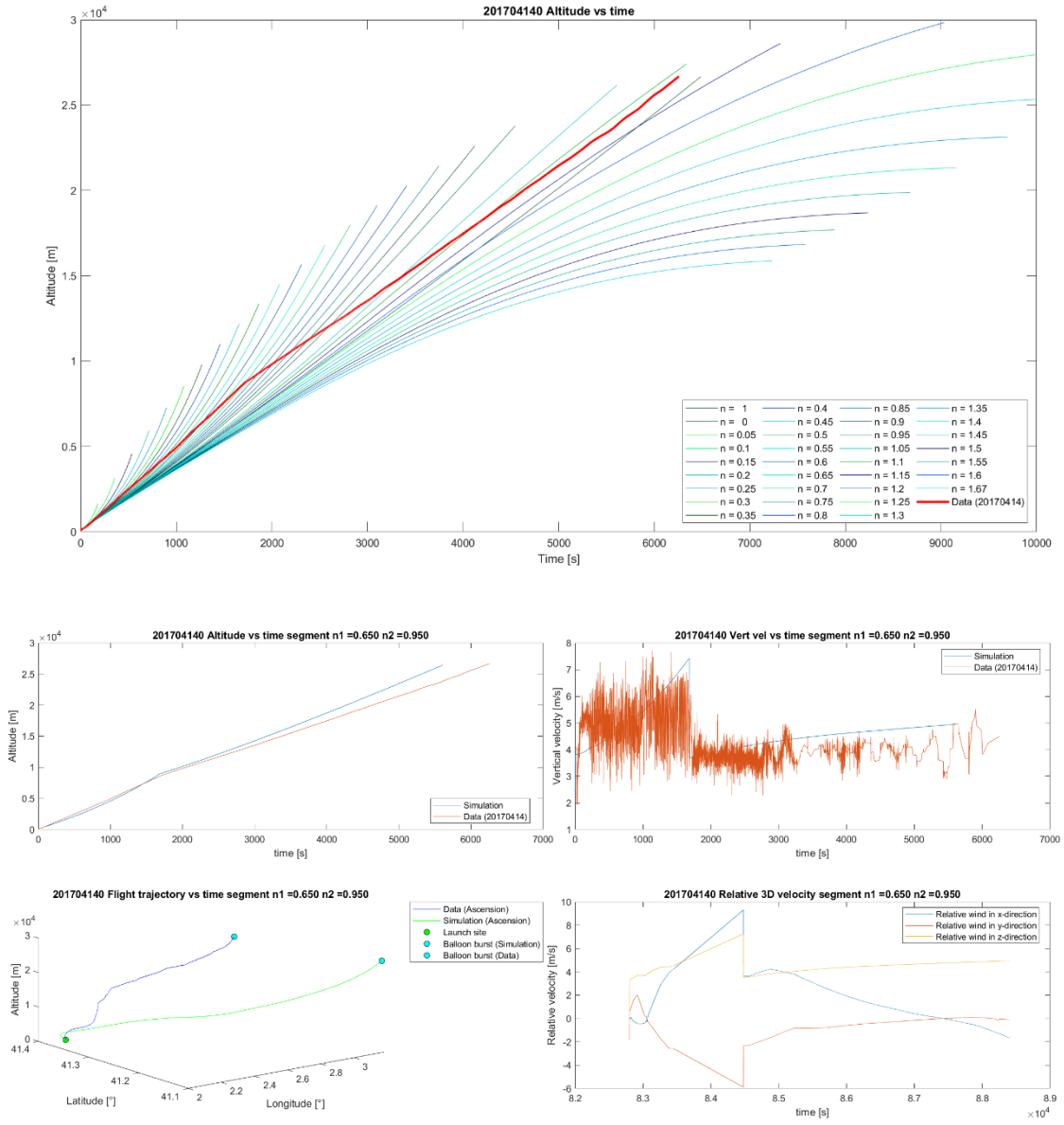


Figure 20 Fit for launch at 14/04/2017 23 UTC

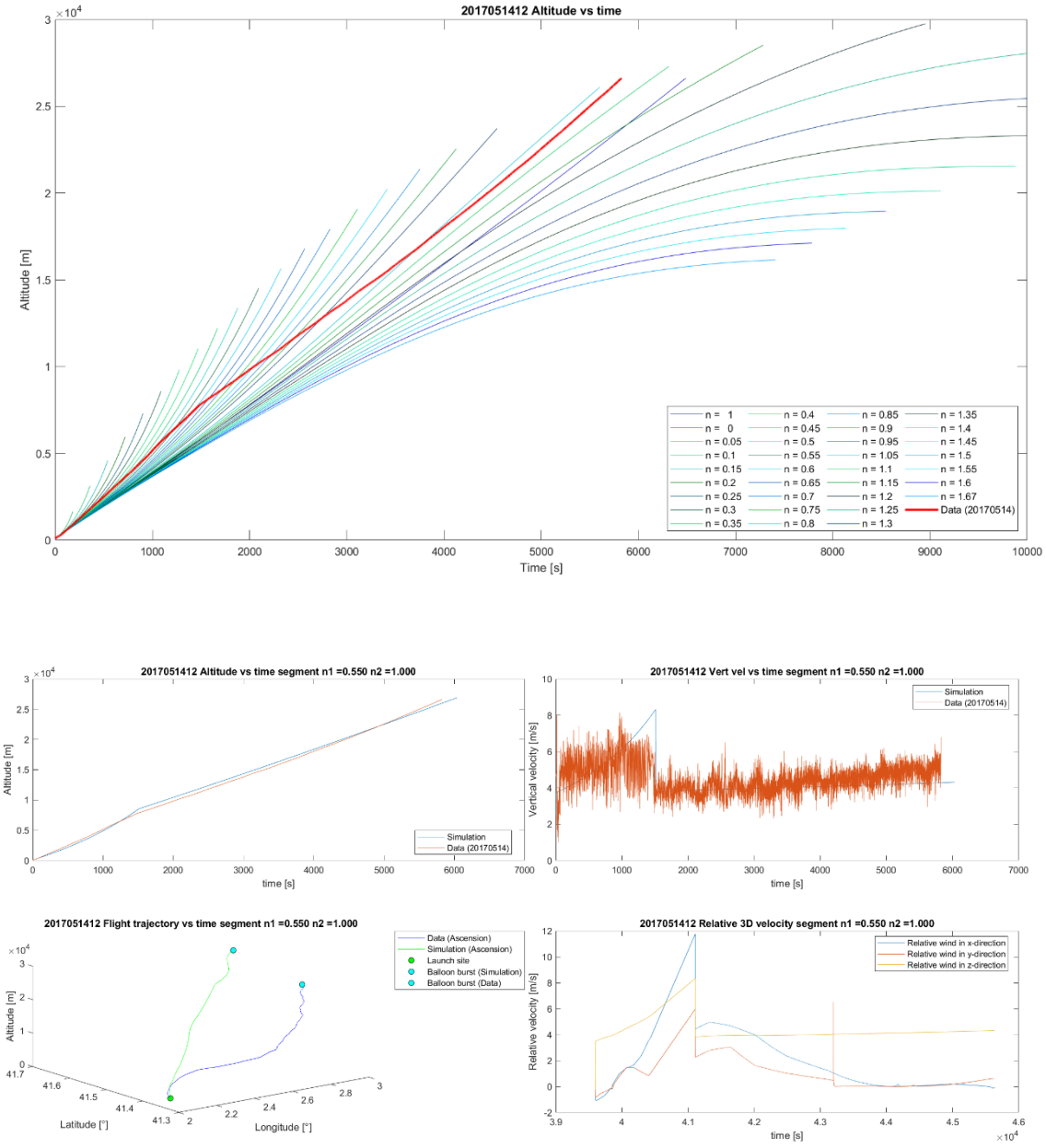


Figure 21 Fit for launch at 14/05/2017 11 UTC

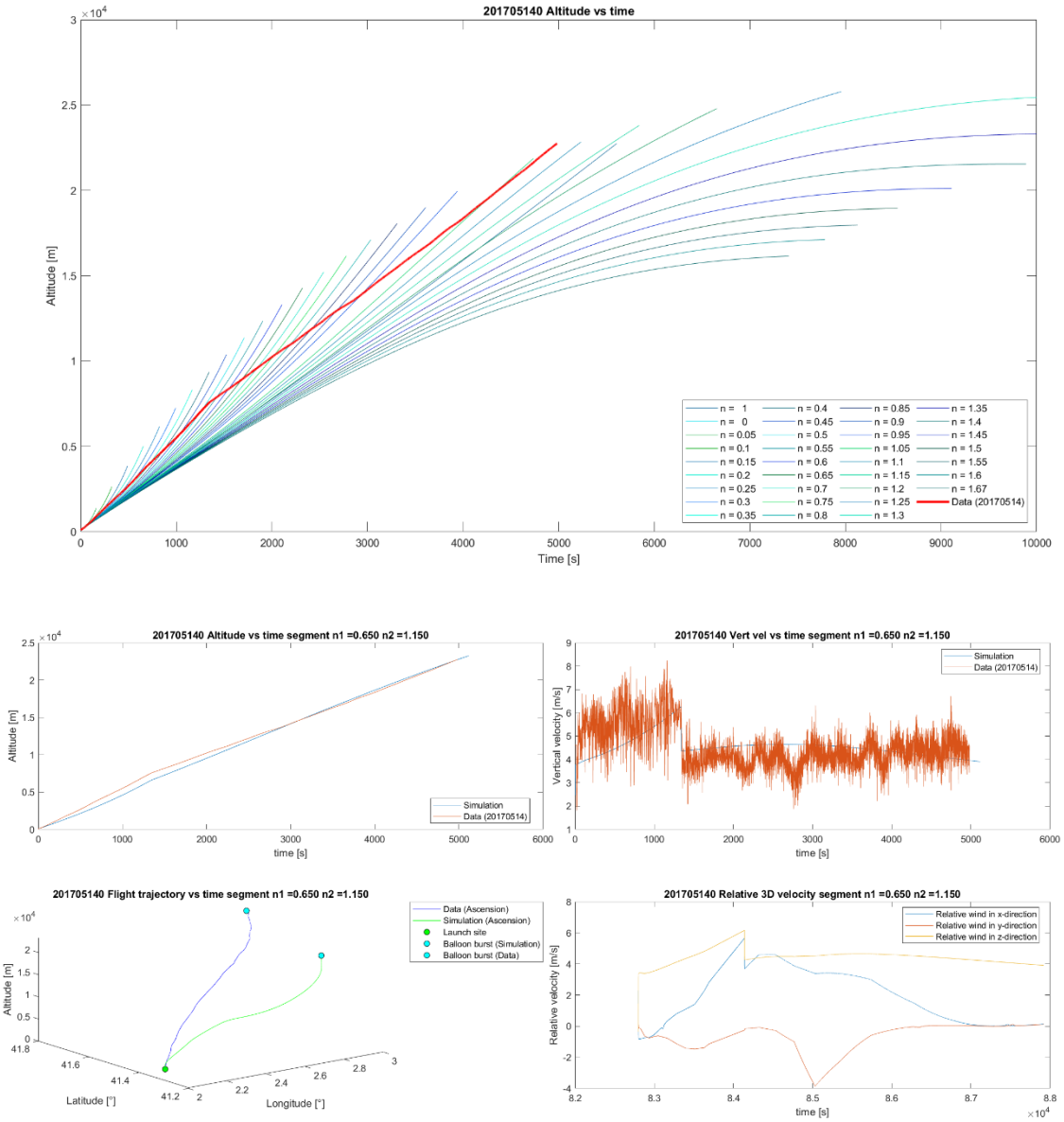


Figure 22 Fit for launch at 14/05/2017 23 UTC

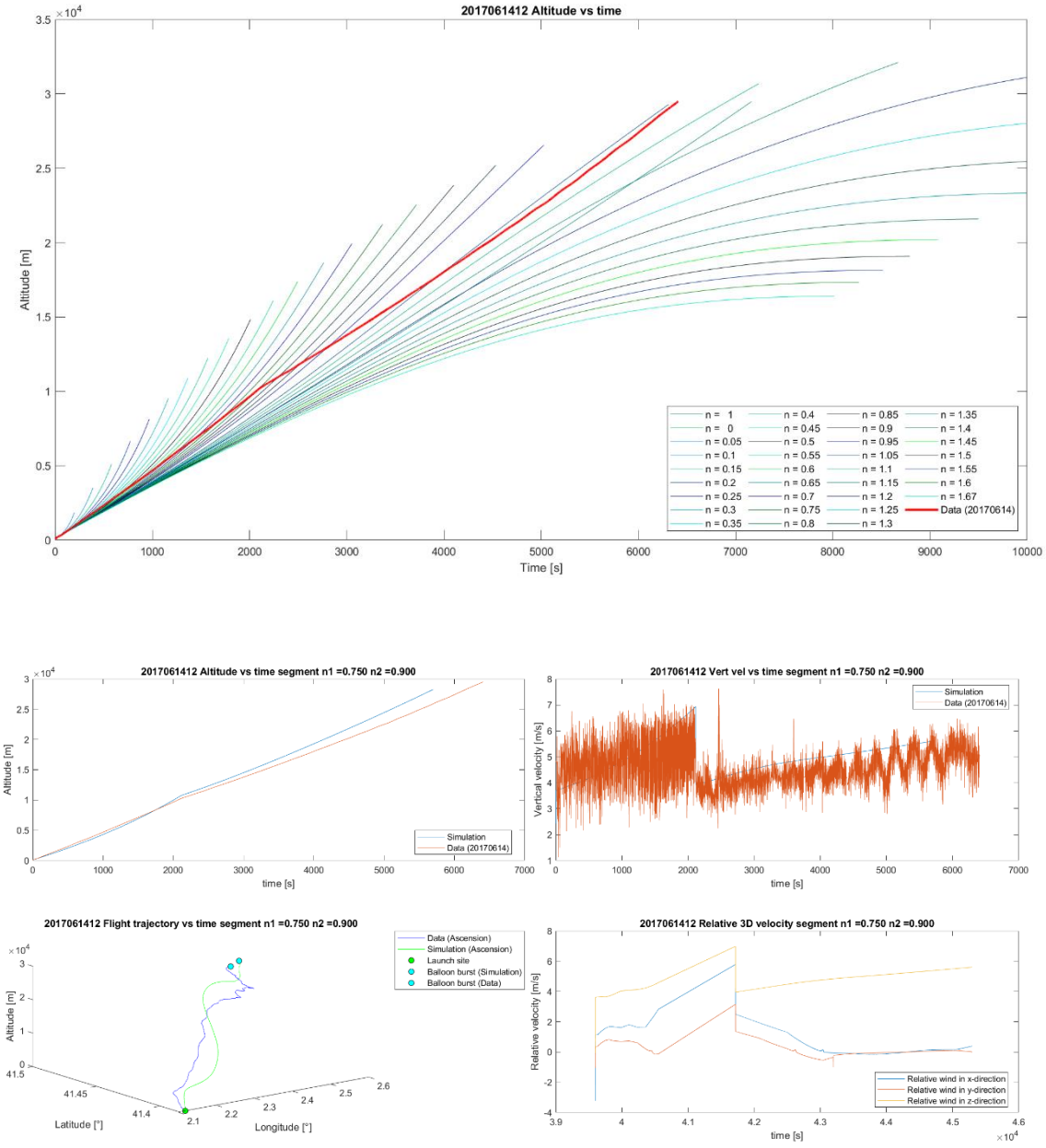


Figure 23 Fit for launch at 14/06/2017 11 UTC

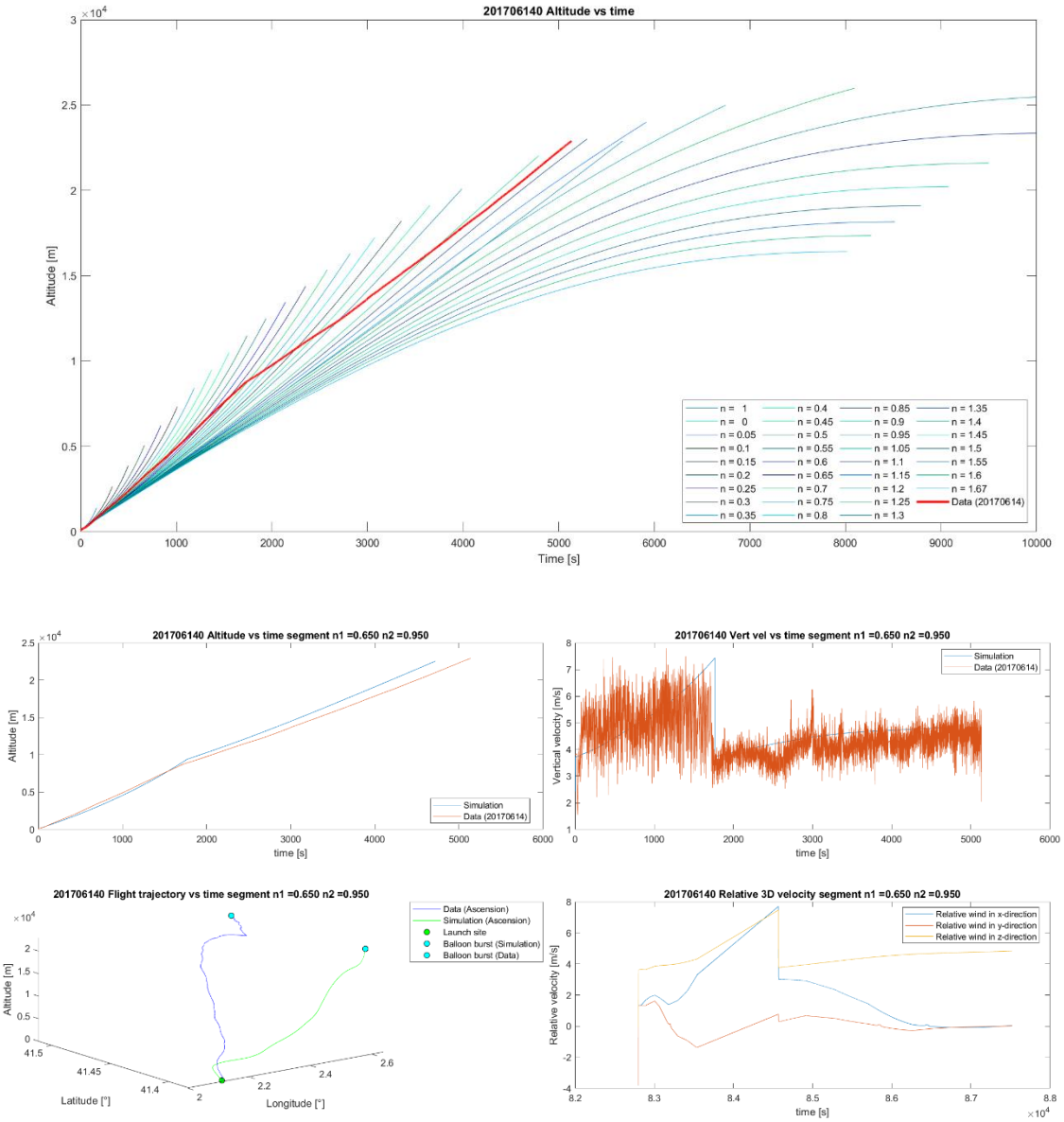


Figure 24 Fit for launch at 14/06/2017 23 UTC

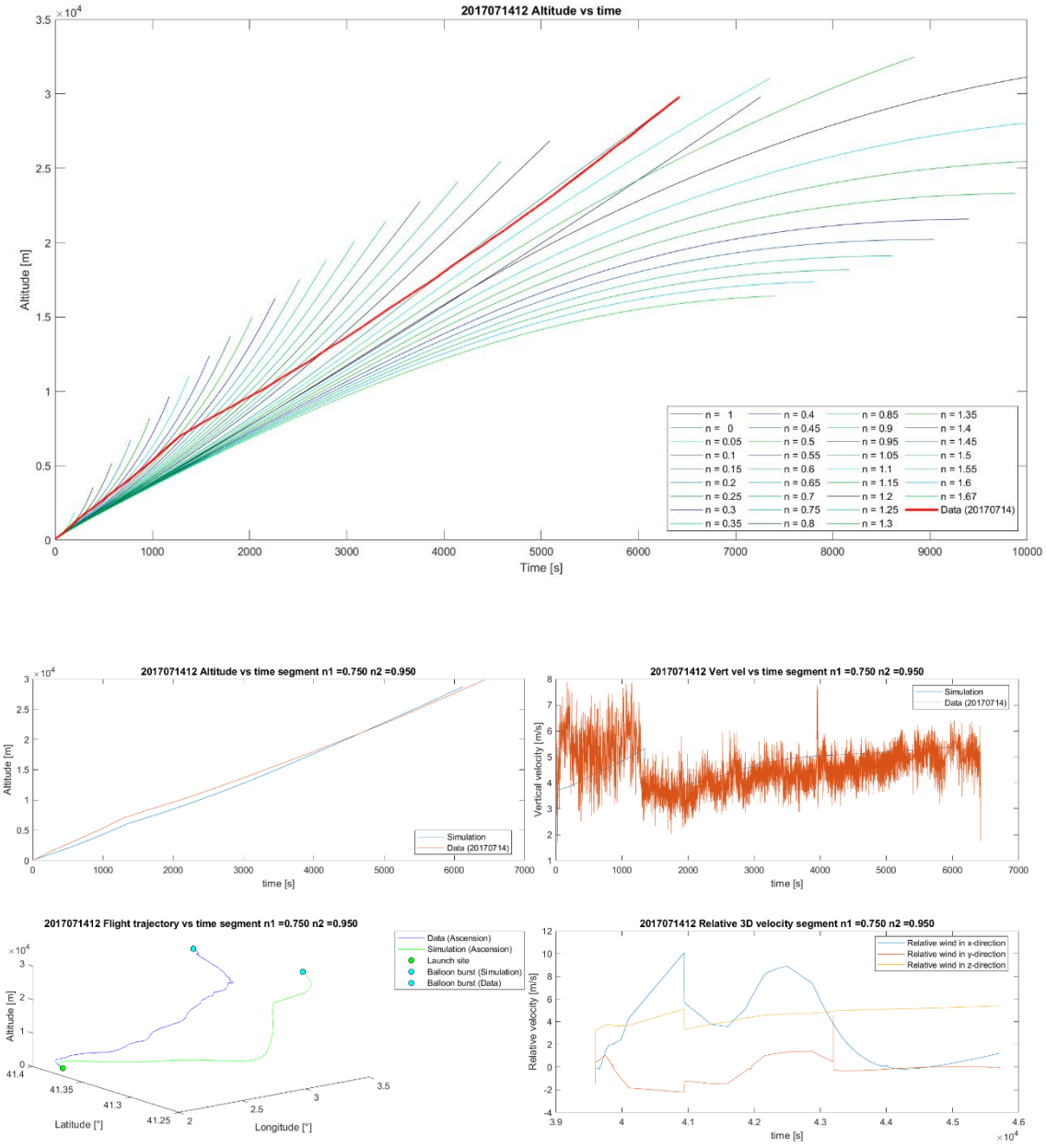


Figure 25 Fit for launch at 14/07/2017 11 UTC

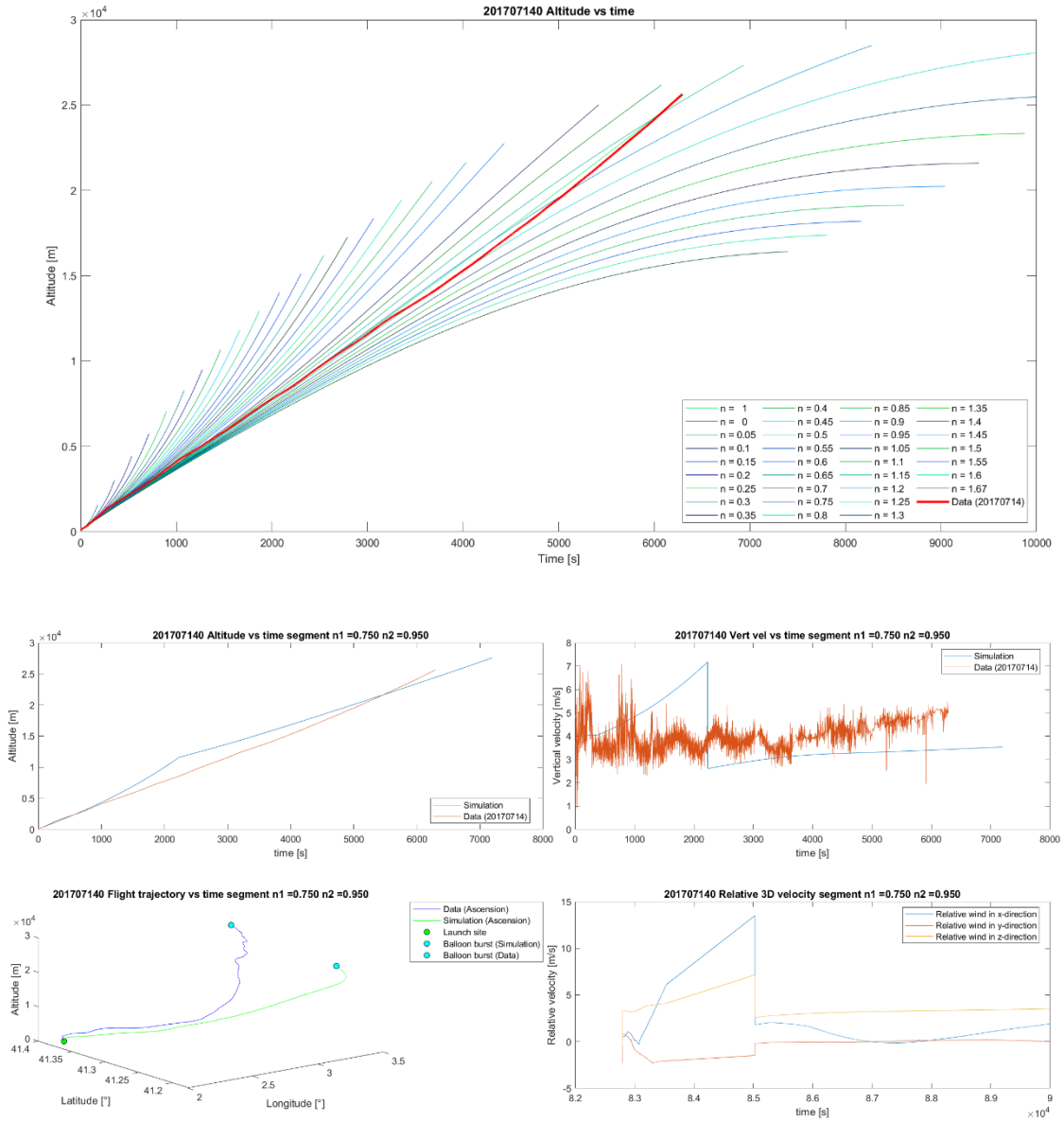


Figure 26 Fit for launch at 14/07/2017 23 UTC

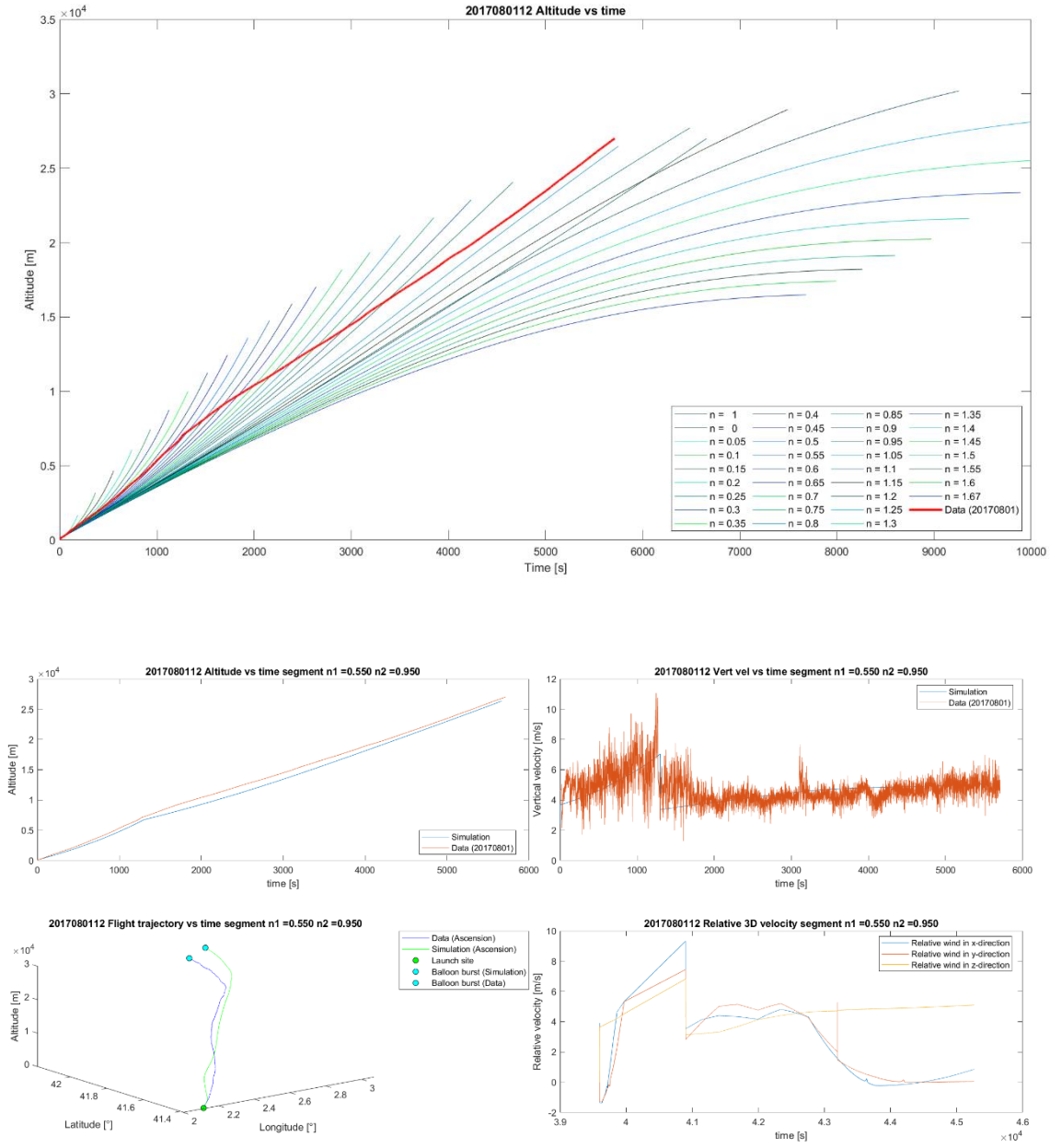


Figure 27 Fit for launch at 01/08/2017 11 UTC

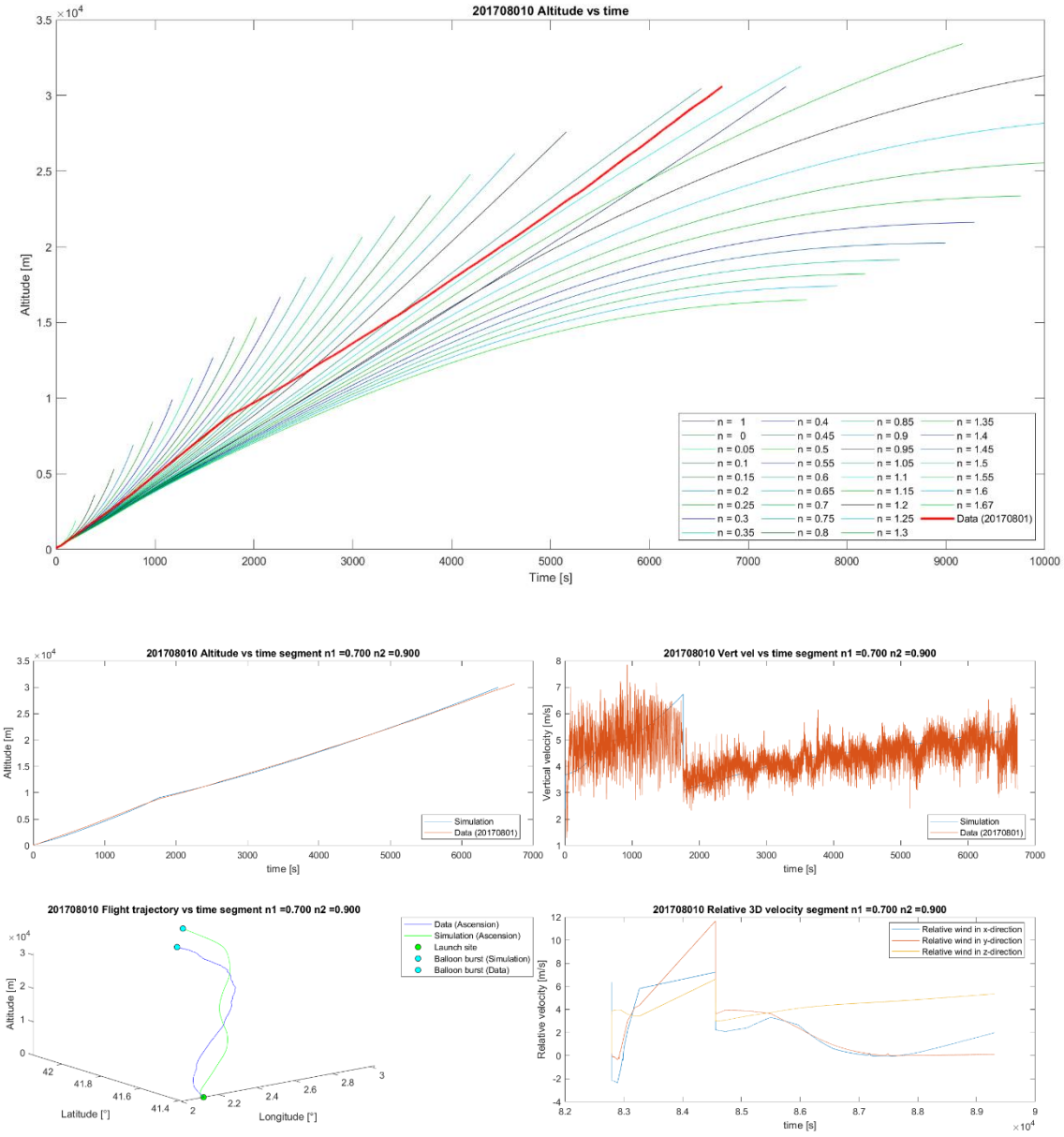


Figure 28 Fit for launch at 01/08/2017 23 UTC

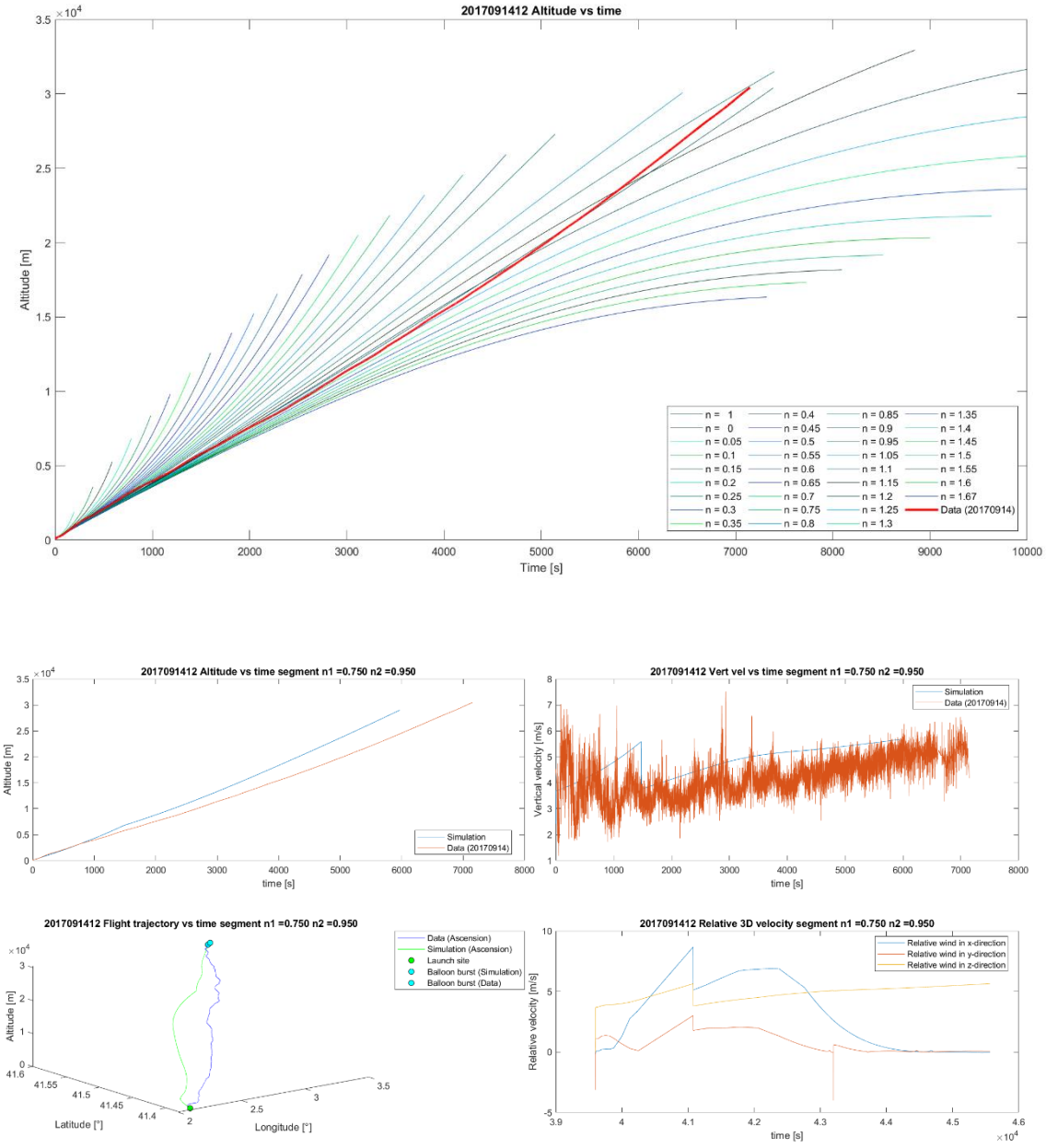


Figure 29 Fit for launch at 14/09/2017 11 UTC

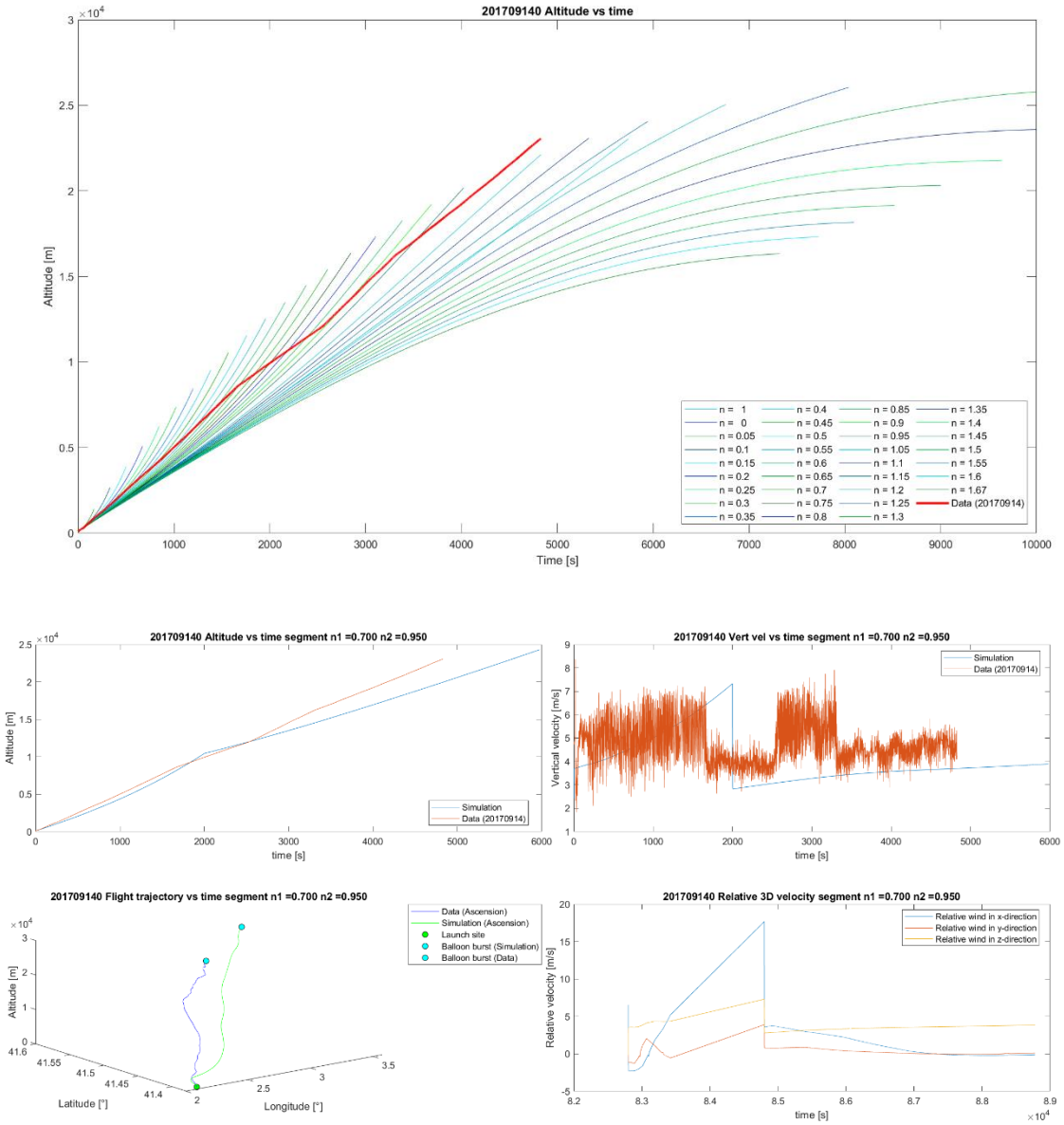


Figure 30 Fit for launch at 14/09/2017 23 UTC

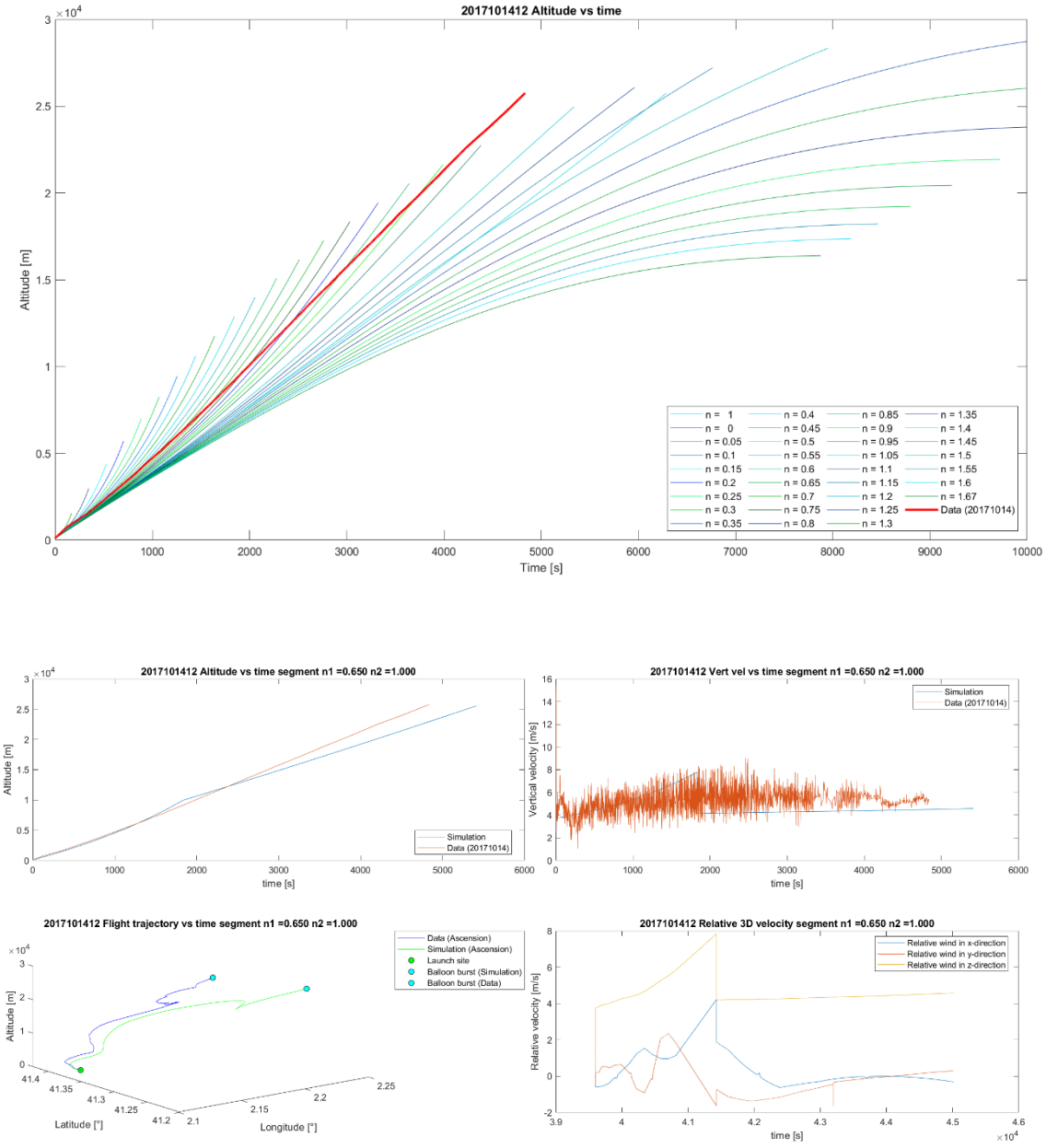


Figure 31 Fit for launch at 14/10/2017 11 UTC

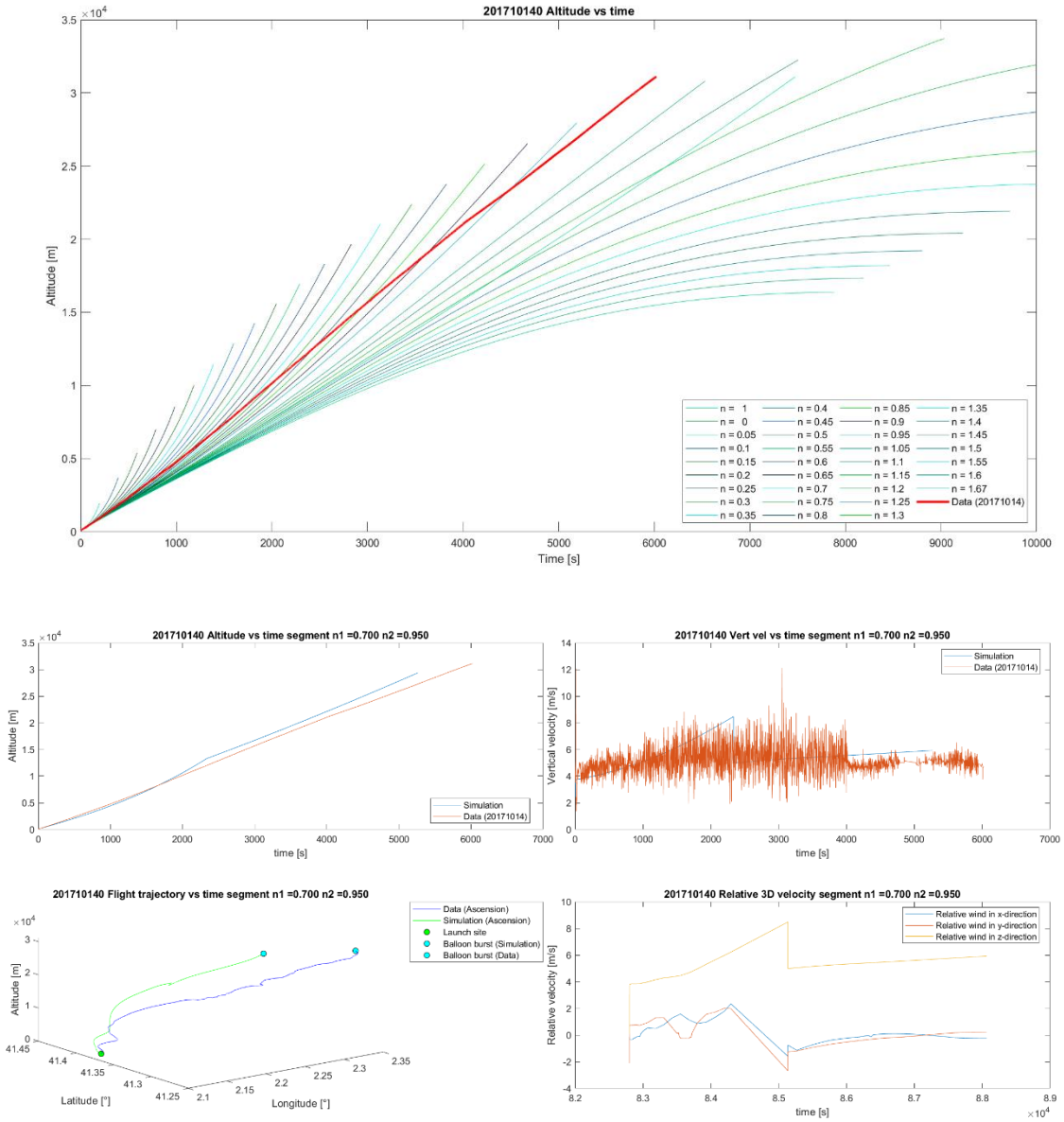


Figure 32 Fit for launch at 14/10/2017 23 UTC

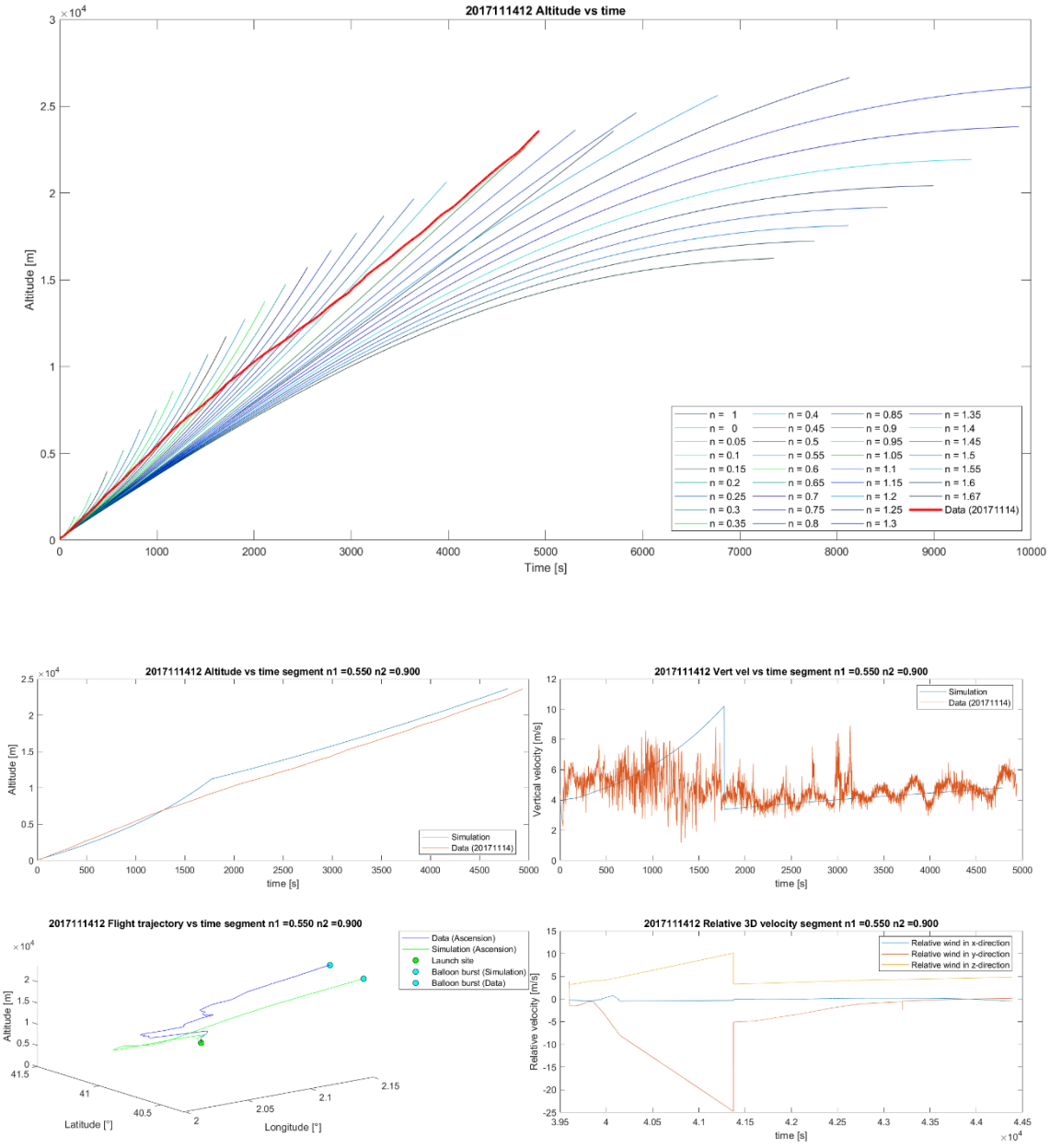


Figure 33 Fit for launch at 14/11/2017 11 UTC

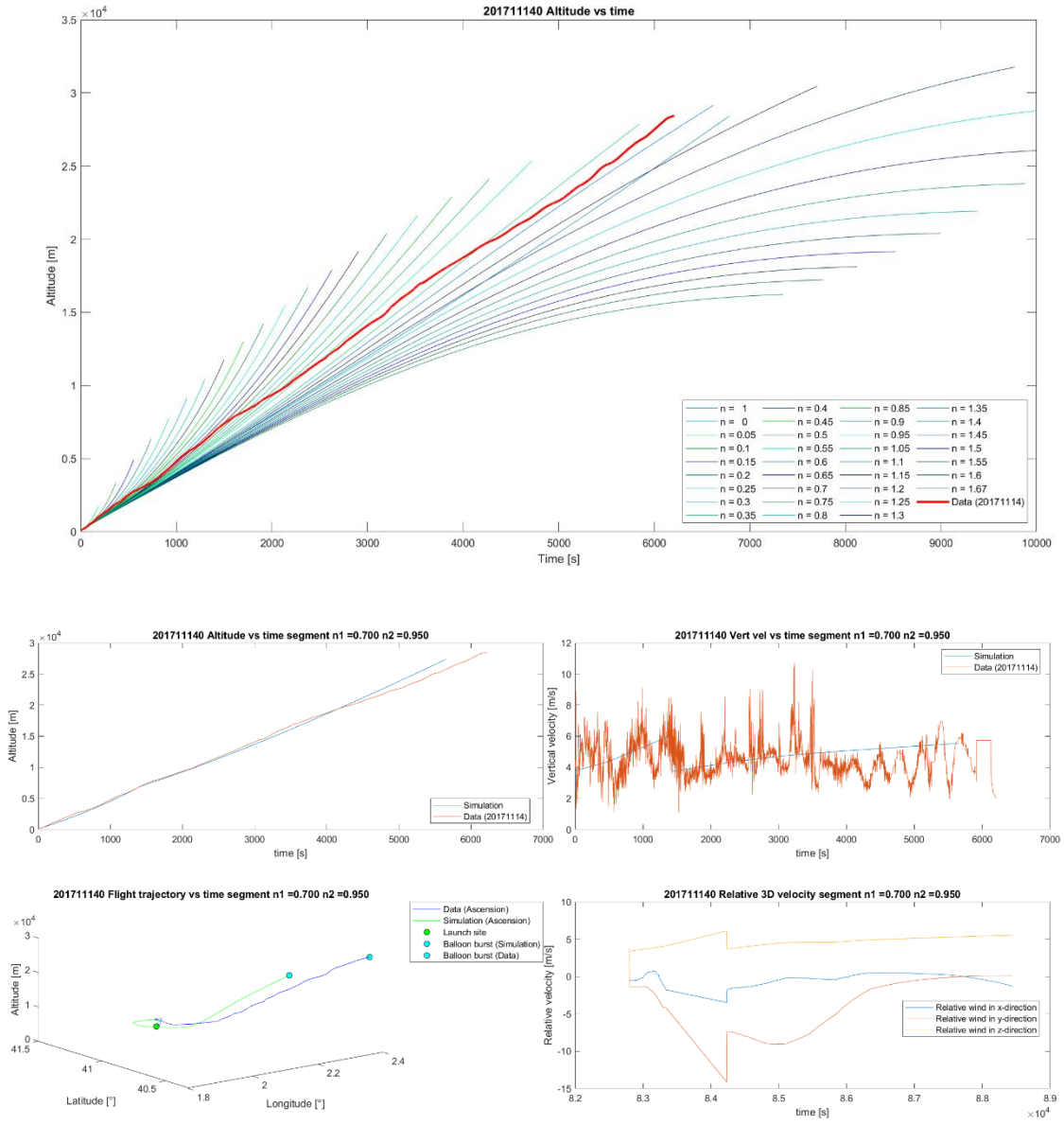


Figure 34 Fit for launch at 14/11/2017 23 UTC

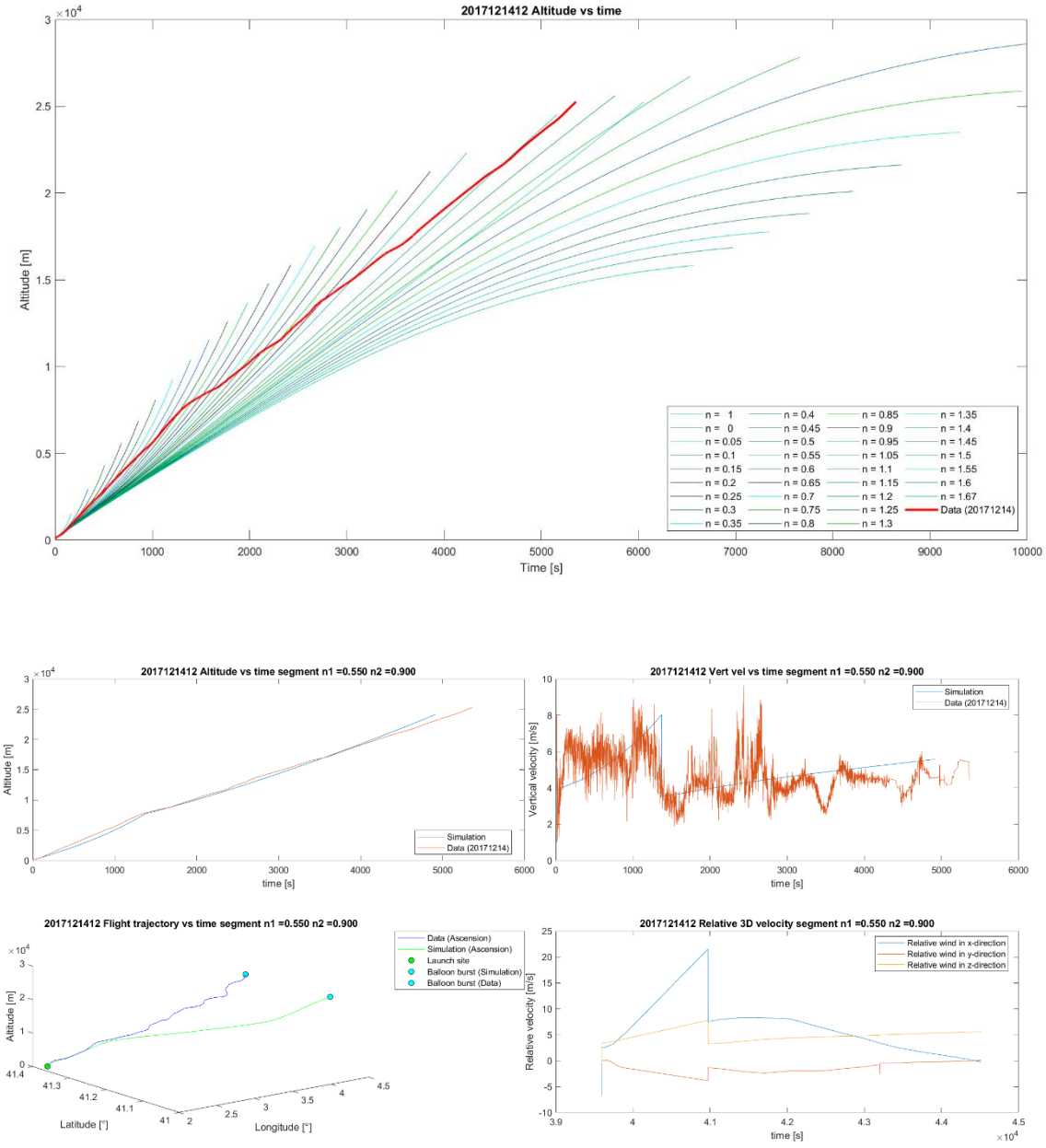


Figure 35 Fit for launch at 14/12/2017 11 UTC

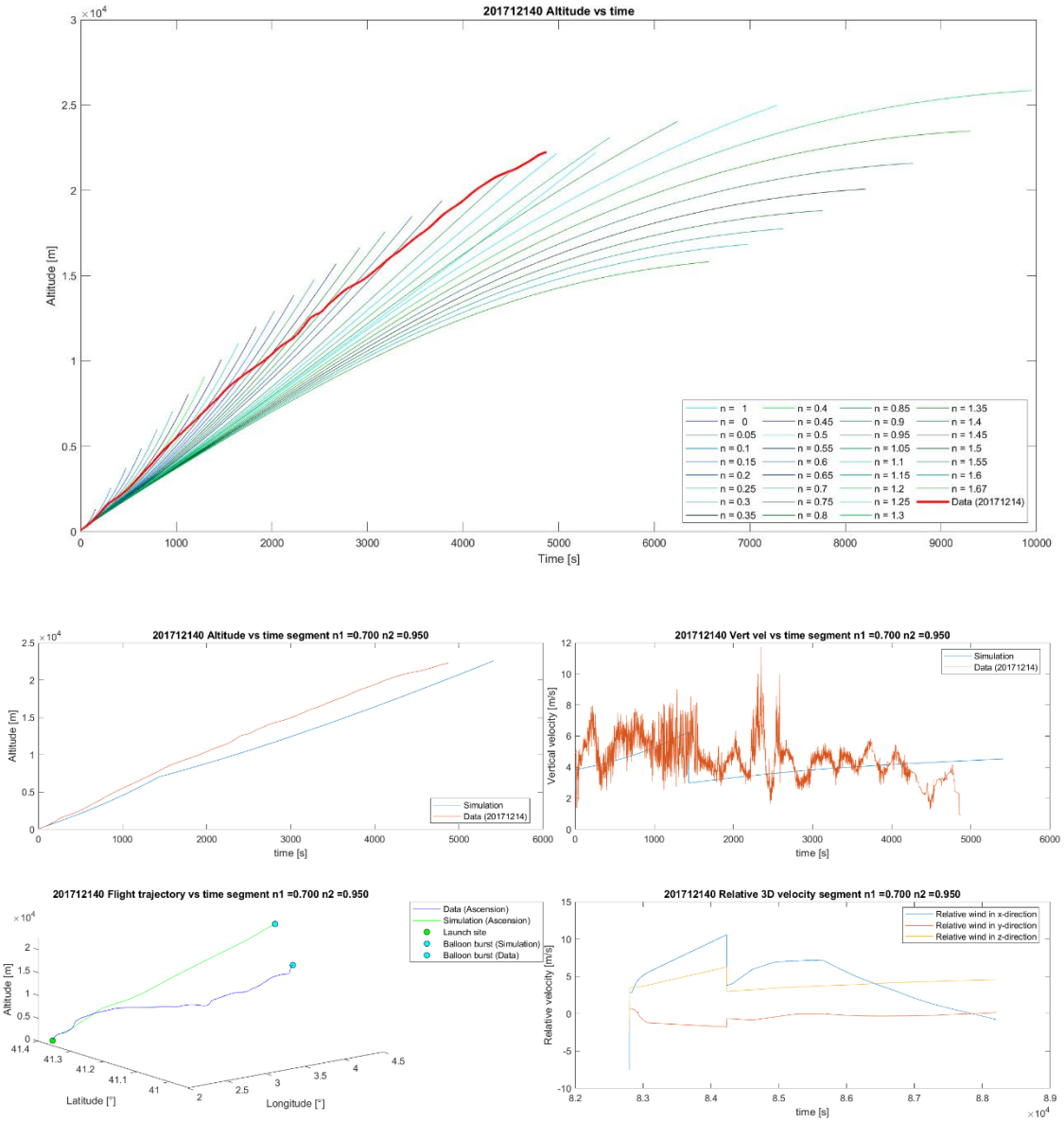


Figure 36 Fit for launch at 14/12/2017 23 UTC

A.4. Simulations error summary

In this chapter the error and values of the position in the final point of the trajectory of the analysed launches per month to do the quarter fitting are presented:

01/01 00	*Data*	*Model*	*Error [km]*
Height [km]	28.57	28.62	0.05
Latitude [º]	41.07	41.32	27.57
Longitude [º]	2.76	3.15	43.22

01/01 12	*Data*	*Model*	*Error [km]*
Height [km]	30.13	30.71	0.58
Latitude [º]	40.81	40.73	8.45
Longitude [º]	2.16	2.25	9.50

14/02 00	*Data*	*Model*	*Error [km]*
Height [km]	20.85	20.71	0.14
Latitude [º]	41.73	41.42	34.49
Longitude [º]	2.25	2.39	15.22

14/02 12	*Data*	*Model*	*Error [km]*
Height [km]	22.60	23.18	0.57
Latitude [º]	41.46	41.47	0.80
Longitude [º]	2.43	2.47	4.50

14/03 00	*Data*	*Model*	*Error [km]*
Height [km]	30.52	29.38	1.14
Latitude [º]	41.40	40.86	60.43
Longitude [º]	1.91	1.68	24.83

14/03 12	*Data*	*Model*	*Error [km]*
Height [km]	23.84	25.04	1.20
Latitude [º]	41.23	41.15	8.23
Longitude [º]	1.83	1.77	6.85

14/04 00	*Data*	*Model*	*Error [km]*
Height [km]	26.69	26.21	0.47
Latitude [º]	41.31	41.10	22.52
Longitude [º]	2.90	3.17	30.15

14/04 12	*Data*	*Model*	*Error [km]*
Height [km]	29.23	28.97	0.26
Latitude [º]	41.28	41.21	8.43
Longitude [º]	3.18	3.22	4.65

14/05 00	*Data*	*Model*	*Error [km]*
Height [km]	22.75	23.27	0.52
Latitude [º]	41.73	41.24	54.17
Longitude [º]	2.97	2.73	26.74

14/05 12	*Data*	*Model*	*Error [km]*
Height [km]	26.62	26.89	0.27
Latitude [º]	41.39	41.64	28.05
Longitude [º]	2.82	2.94	13.61

14/06 00	*Data*	*Model*	*Error [km]*
Height [km]	22.54	23.65	1.11
Latitude [º]	41.39	41.38	1.31
Longitude [º]	2.61	2.56	6.08

14/06 12	*Data*	*Model*	*Error [km]*
Height [km]	29.49	28.27	1.22
Latitude [º]	41.44	41.46	1.44
Longitude [º]	2.43	2.50	7.21

14/07 00	*Data*	*Model*	*Error [km]*
Height [km]	25.64	27.59	1.95
Latitude [º]	41.38	41.17	22.49
Longitude [º]	3.36	3.14	24.80

14/07 12	*Data*	*Model*	*Error [km]*
Height [km]	29.80	28.72	1.08
Latitude [º]	41.38	41.29	9.06
Longitude [º]	3.08	3.32	26.26

01/08 00	*Data*	*Model*	*Error [km]*
Height [km]	30.60	29.94	0.66
Latitude [º]	41.93	42.14	22.53
Longitude [º]	2.54	2.78	26.52

01/08 12	*Data*	*Model*	*Error [km]*
Height [km]	27.00	26.33	0.67
Latitude [º]	42.07	42.17	11.75
Longitude [º]	2.74	2.94	22.10

14/09 00	*Data*	*Model*	*Error [km]*
Height [km]	23.06	24.33	1.27
Latitude [º]	41.52	41.59	7.52
Longitude [º]	2.92	3.57	72.58

14/09 12	*Data*	*Model*	*Error [km]*
Height [km]	30.41	28.98	1.43
Latitude [º]	41.57	41.58	1.10
Longitude [º]	3.25	3.28	3.84

14/10 00	*Data*	*Model*	*Error [km]*
Height [km]	31.11	29.38	1.73
Latitude [º]	41.26	41.31	5.92
Longitude [º]	2.32	2.26	7.41

14/10 12	*Data*	*Model*	*Error [km]*
Height [km]	25.76	25.55	0.21
Latitude [º]	41.33	41.24	9.46
Longitude [º]	2.19	2.22	3.35

14/11 00	*Data*	*Model*	*Error [km]*
Height [km]	28.44	27.35	1.08
Latitude [º]	40.31	40.32	0.67
Longitude [º]	2.36	2.11	27.16

14/11 12	*Data*	*Model*	*Error [km]*
Height [km]	23.59	23.67	0.08
Latitude [º]	40.64	40.31	36.61
Longitude [º]	2.15	2.14	0.61

14/12 00	*Data*	*Model*	*Error [km]*
Height [km]	22.24	22.58	0.34
Latitude [º]	40.95	41.22	29.28
Longitude [º]	3.40	4.26	96.40

14/12 12	*Data*	*Model*	*Error [km]*
Height [km]	25.26	24.13	1.13
Latitude [º]	41.23	41.04	20.91
Longitude [º]	3.96	4.17	23.26

Table 2 Fit simulations error at the end of the trajectory

Annex B

In this Annex, the different codes used to obtain and arrange data, run the simulation and the verification of the model are presented.

B.1. Script Main_Data_Import

```
% THis script is used to import data in the launches log files into Matlab
% as a .mat database in order to read it easily to perform the validation.

% clear; clc;
loaded = 1; % To avoid reloads
if loaded == 0
    %% Delete non useful files
    cd 'GLOBUS_SONDA_UB_RAOB_99-17'
    delete .*.*;
    % Year range
    for year = 1999:2017
        cd(sprintf('%d',year));
        delete .*.*;
        delete .*.txt;
        % Month range
        for month = 1:12
            if month <10
                cd(sprintf('0%d',month));
            else
                cd(sprintf('%d',month));
            end
            delete .*.txt;
            cd ..
        end
    cd ..
    end
    cd ..

    %% Data import
    year_ini=2012;
    year_fin =2017;
    % Auxiliar initializations
    DATB.lch_year = double.empty(0,1);
    DATB.lch_month = double.empty(0,1);
    DATB.lch_day = double.empty(0,1);
    DATB.lch_hour = double.empty(0,1);
    DATB.ftr_time = double.empty(0,1);          % Temps [s]
    DATB.ftr_alt = double.empty(0,1);           % Altura [m]
    DATB.ftr_pres = double.empty(0,1);          % Pressió [hPa/mbar]
    DATB.ftr_temp = double.empty(0,1);          % Temperatura [ºC]
    DATB.ftr_hum = double.empty(0,1);           % Humitat relativa [%]
    DATB.ftr_DP = double.empty(0,1);            % Temperatura del Punt de Rosada
    DATB.ftr_WF = double.empty(0,1);            % Velocitat del vent (Wind Force) [m/s o kts]
    DATB.ftr_WD = double.empty(0,1);            % Direcció del vent [°]
    DATB.ftr_VEF = double.empty(0,1);           % Component de vent horizontal u  [m/s o kts]
    DATB.ftr_VNF = double.empty(0,1);           % Component de vent horizontal v  [m/s o kts]
    DATB.ftr_LAT = double.empty(0,1);           % Latitud [°]
    DATB.ftr_LON = double.empty(0,1);           % Longitud [°]
    % Copy the import function inside subfolders
    copyfile importfile.m GLOBUS_SONDA_UB_RAOB_99-17
    cd 'GLOBUS_SONDA_UB_RAOB_99-17'
```

```
    % Year range
    for year = year_ini:year_fin
        copyfile('importfile.m',sprintf('%d',year))
        cd(sprintf('%d',year));
        for month = 1:12
            if month <10
                copyfile('importfile.m',sprintf('0%d',month))
                cd(sprintf('0%d',month));
                list=dir('*.txt');
            else
                copyfile('importfile.m',sprintf('%d',month))
                cd(sprintf('%d',month));
                list=dir('*.txt');
            end

            for n = 1:length(list)

[ftr_time,ftr_alt,ftr_pres,ftr_temp,ftr_hum,ftr_DP,ftr_WF,ftr_WD,ftr_VEF,ftr_VNF,ftr_LAT,ftr_LON]
= importfile(list(n).name, 2, inf);
                lch_year = zeros(length(ftr_time),1)+str2double(year);
                lch_month = zeros(length(ftr_time),1)+str2double(month);
                lch_day = zeros(length(ftr_time),1) + str2double(list(n).name(5:6));
                lch_hour = zeros(length(ftr_time),1) + str2double(list(n).name(7:8));
                DATB.lch_year = vertcat(DATB.lch_year,lch_year);
                DATB.lch_month = vertcat(DATB.lch_month,lch_month);
                DATB.lch_day = vertcat(DATB.lch_day,lch_day);
                DATB.lch_hour = vertcat(DATB.lch_hour,lch_hour);
                DATB.ftr_time = vertcat(DATB.ftr_time,ftr_time);
                DATB.ftr_alt =  vertcat(DATB.ftr_alt,ftr_alt);
                DATB.ftr_pres = vertcat(DATB.ftr_pres,ftr_pres);
                DATB.ftr_temp = vertcat(DATB.ftr_temp,ftr_temp);
                DATB.ftr_hum = vertcat(DATB.ftr_hum,ftr_hum);
                DATB.ftr_DP = vertcat(DATB.ftr_DP,ftr_DP);
                DATB.ftr_WF = vertcat(DATB.ftr_WF,ftr_WF);
                DATB.ftr_WD = vertcat(DATB.ftr_WD,ftr_WD);
                DATB.ftr_VEF = vertcat(DATB.ftr_VEF,ftr_VEF);
                DATB.ftr_VNF = vertcat(DATB.ftr_VNF,ftr_VNF);
                DATB.ftr_LAT = vertcat(DATB.ftr_LAT,ftr_LAT);
                DATB.ftr_LON = vertcat(DATB.ftr_LON,ftr_LON);
            end
            delete 'importfile.m';
            cd ..
        end
        delete 'importfile.m';
        cd ..
    end
    delete 'importfile.m'
    cd ..
    save('DATB.mat', 'DATB', '-v7.3')
end

% Organize by years
for year = 2012:2017
    for month =1:12
        [ DATByyyymm ] = rearrange_IN( DATB, year, month );
    end
end
```

B.1.1. Function importfile

```
function
[ftr_time,ftr_alt,ftr_pres,ftr_temp,ftr_hum,ftr_DP,ftr_WF,ftr_WD,ftr_VEF,ftr_VNF,ftr_LAT,ftr_LON]
= importfile(filename, startRow, endRow)
%IMPORTFILE Import numeric data from a text file as column vectors.
%
[FTR_TIME,FTR_ALT,FTR_PRES,FTR_TEMP,FTR_HUM,FTR_DP,FTR_WF,FTR_WD,FTR_V
EF,FTR_VNF,FTR_LAT,FTR_LON]
%   = IMPORTFILE(FILENAME) Reads data from text file FILENAME for the
%   default selection.
%
%
[FTR_TIME,FTR_ALT,FTR_PRES,FTR_TEMP,FTR_HUM,FTR_DP,FTR_WF,FTR_WD,FTR_V
EF,FTR_VNF,FTR_LAT,FTR_LON]
%   = IMPORTFILE(FILENAME, STARTROW, ENDROW) Reads data from rows STARTROW
%   through ENDROW of text file FILENAME.
%
% Example:
%
[ftr_time,ftr_alt,ftr_pres,ftr_temp,ftr_hum,ftr_DP,ftr_WF,ftr_WD,ftr_VEF,ftr_VNF,ftr_LAT,ftr_LON]
= importfile('12010100.txt',2, 2237);
%
%    See also TEXTSCAN.

% Auto-generated by MATLAB on 2019/01/31 17:57:33

%% Initialize variables.
delimiter = '\t';
if nargin<=2
startRow = 2;
endRow = inf;
end

%% Format string for each line of text:
%   column1: double (%f)
%   column2: double (%f)
%   column3: double (%f)
%   column4: double (%f)
%   column5: double (%f)
%   column6: double (%f)
%   column7: double (%f)
%   column8: double (%f)
%   column9: double (%f)
%   column10: double (%f)
%   column11: double (%f)
%   column12: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%f%f%f%f%f%f%f%f%f%f%f%f%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');
%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1, 'Delimiter', delimiter,
'EmptyValue' ,NaN,'HeaderLines', startRow(1)-1, 'ReturnOnError', false);
for block=2:length(startRow)
```

```
frewind(fileID);
dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-startRow(block)+1, 'Delimiter',
delimiter, 'EmptyValue' ,NaN,'HeaderLines', startRow(block)-1, 'ReturnOnError', false);
for col=1:length(dataArray)
dataArray{col} = [dataArray{col};dataArrayBlock{col}];
end
end

%% Close the text file.
fclose(fileID);
%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Allocate imported array to column variable names
ftr_time = dataArray{:, 1};
ftr_alt = dataArray{:, 2};
ftr_pres = dataArray{:, 3};
ftr_temp = dataArray{:, 4};
ftr_hum = dataArray{:, 5};
ftr_DP = dataArray{:, 6};
ftr_WF = dataArray{:, 7};
ftr_WD = dataArray{:, 8};
ftr_VEF = dataArray{:, 9};
ftr_VNF = dataArray{:, 10};
ftr_LAT = dataArray{:, 11};
ftr_LON = dataArray{:, 12};
```

B.1.2. Function rearrange_IN(

```
function [ DATByyyymm ] = rearrange_IN( DATB, year, month )
%rearrange_IN Rearranges data input by year and month
%   Inputs:
%     DATB          :  Baloon launch database
%     year          :  Launch year
%     month         :  Launch month
%   Outputs:
%     DATByyyymm    :  Baloon launch for a given month and year
samples=DATB.lch_year==year& DATB.lch_month==month;
DATByyyymm.lch_year = DATB.lch_year(samples);
DATByyyymm.lch_month = DATB.lch_month(samples);
DATByyyymm.lch_day = DATB.lch_day(samples);
DATByyyymm.lch_hour = DATB.lch_hour(samples);
DATByyyymm.ftr_time = DATB.ftr_time(samples);
DATByyyymm.ftr_alt = DATB.ftr_alt(samples);
DATByyyymm.ftr_pres = DATB.ftr_pres(samples);
DATByyyymm.ftr_temp = DATB.ftr_temp(samples);
DATByyyymm.ftr_hum = DATB.ftr_hum(samples);
DATByyyymm.ftr_DP = DATB.ftr_DP(samples);
DATByyyymm.ftr_WF = DATB.ftr_WF(samples);
DATByyyymm.ftr_WD = DATB.ftr_WD(samples);
DATByyyymm.ftr_VEF = DATB.ftr_VEF(samples);
DATByyyymm.ftr_VNF = DATB.ftr_VNF(samples);
DATByyyymm.ftr_LAT = DATB.ftr_LAT(samples);
DATByyyymm.ftr_LON = DATB.ftr_LON(samples);
if month <10
   filename = sprintf('DATB%d0%d.mat',year,month);
else
```

```
    filename = sprintf('DATB%d%d.mat',year,month);
end
% fldnm = sprintf('DATB%d0%d',year,month);
% DATByyyymm.(fldnm) = DATBsamples;
cd 'GLOBUS_SONDA_UB_RAOB_COMPACT'
save(filename, 'DATByyyymm')
cd ..
end
```

B.2. Script process_grib2

```
% This script downloads NOAA weather data for a specific date. It also
% extracts and saves the wind data from the downloaded grb2-files
clc; clear; close all

% Use forecast (mode 1) or analysis (mode 2) data
mode = 2;
% Delete grb2-file after extracting and saving wind data? (del = 1: yes,
% del = 2: no)
del = 1;

%% Date
year = 2017;
yyyy = sprintf('%d',year);
month = 12;
if month <10
    mm = sprintf('0%d',month);
else
    mm = sprintf('%d',month);
end
day = eomday(year, month);
if day <10
    dd = sprintf('0%d',day);
else
    dd = sprintf('%d',day);
end
date = [yyyy,mm,dd];

%% Download grib2 files from NOAA
for i=1:4
    if i == 1
        time = '0000';
    elseif i == 2
        time = '0600';
    elseif i == 3
        time = '1200';
    elseif i == 4
        time = '1800';
    end
    fctime = '000'; % Forecast time available in 3h steps with 3 digits
    if exist(['simulation/wind_data/wind_',date,'_',time,'.mat'], 'file') ~= 2
        % Forecast
        if mode == 1
            dataname = ['gfs_4_',date,'_',time,'_',fctime,'.grb2'];
            % In case the normal link is not accessible, a second one will be tried
            try
                url = ['https://nomads.ncdc.noaa.gov/data/gfs4/',yyyy,mm,'/',date,'/',dataname];
                filename = ([date,'_',time,'.grb2']);
                cd ('D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data codes\GRIB2\');
```

```
            websave(filename,url);
        catch
            url = ['https://www.ncei.noaa.gov/thredds/fileServer/gfs-004-files/',...
                yyyy,mm,'/',date,'/',dataname];
            filename = ([date,'_',time,'.grb2']);
            cd ('D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data codes\GRIB2\');
            websave(filename,url);
        end
        % Analysis
    elseif mode == 2
        dataname = ['gfsanl_4_',date,'_',time,'_',fctime,'.grb2'];
        % In case the normal link is not accessible, a second one will be tried
        try
            url = ['https://nomads.ncdc.noaa.gov/data/gfsanl/',yyyy,mm,'/',date,'/',dataname];
            filename = ([date,'_',time,'.grb2']);
            ('D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data codes\GRIB2\');
            websave(filename,url);
        catch
            url = ['https://www.ncei.noaa.gov/thredds/fileServer/gfs-g4-anl-files/',...
                yyyy,mm,'/',date,'/',dataname];
            filename = ([date,'_',time,'.grb2']);
            ('D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data codes\GRIB2\');
            websave(filename,url);
        end
    end

    %% Handling grib2 data
    % "Activate nctoolbox"
    cd ('D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data codes\nctoolbox-1.1.3\');
    setup_nctoolbox
    % Get data
    url = ['D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data
codes\GRIB2\',date,'_',time,'.grb2'];
    nc = ncgeodataset(url);
    % Find the fitting pressure vectors
    pressure_u_v = nc.dimensions('u-component_of_wind_isobaric');
    pressure_vert = nc.dimensions('Vertical_velocity_pressure_isobaric');
    % Get wind components
    size_p = nc.size(pressure_u_v{2});
    size_p_vert = nc.size(pressure_vert{2});
    for ii=1:size_p
        wind.wind_u{ii} = squeeze(nc{'u-component_of_wind_isobaric'}(1,ii,:,:));
        wind.wind_v{ii} = squeeze(nc{'v-component_of_wind_isobaric'}(1,ii,:,:));
        % Corresponding grid
        wind.grid_u{ii} = nc{'u-component_of_wind_isobaric'}(1,ii,:,:).grid;
        wind.grid_v{ii} = nc{'v-component_of_wind_isobaric'}(1,ii,:,:).grid;
        % Plot
        %       figure
        %       pcolorjw(wind.grid_u{i}.lon,wind.grid_u{i}.lat,wind.wind_u{i})
        %       colorbar
        %       figure
        %       pcolorjw(wind.grid_v{i}.lon,wind.grid_v{i}.lat,wind.wind_v{i})
        %       colorbar
    end
    for ii=1:size_p_vert
        wind.wind_vert{ii} = squeeze(nc{'Vertical_velocity_pressure_isobaric'}(1,ii,:,:));
        % Corresponding grid
        wind.grid_vert{ii} = nc{'Vertical_velocity_pressure_isobaric'}(1,ii,:,:).grid;
        %     % Plot
```

```
            % figure;pcolor(wind.grid_u{1}.lon,wind.grid_u{1}.lat,wind.wind_u{20});hold on;shading
interp;
            %     pcolorjw(wind.grid_vert{i}.lon,wind.grid_vert{i}.lat,wind.wind_vert{i})
            %     colorbar
        end
        wind.p = nc{pressure_u_v{2}}(:);
        wind.p_vert = nc{pressure_vert{2}}(:);
        % Save wind files in easy readable matlab format
        save(['wind_',date,'_',time,'.mat'],'wind');
        % Save at wished location
        source = ['wind_',date,'_',time,'.mat'];
        destination = 'D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data
codes\simulation\wind_data';
        movefile(source,destination)
    else
    end
end
if del == 1
    cd ..;
    delete *.grb2;
    delete *.grb2.html;
    delete *.grb2.gbx9;
    delete *.grb2.ncx;
end
```

B.3. Script process_grib2_download

```
% This script downloads NOAA weather data for a specific date.
clc; clear; close all

% Use forecast (mode 1) or analysis (mode 2) data
mode = 1;

%% Date
year = 2017;
month = 12;
day = 27;
yyyy = sprintf('%d',year);
if month <10
    mm = sprintf('0%d',month);
else
    mm = sprintf('%d',month);
end
if day <10
    dd = sprintf('0%d',day);
else
    dd = sprintf('%d',day);
end
date = [yyyy,mm,dd];

    %% Download grib2 files from NOAA
    for i=1:4
        if i == 1
            time = '0000';
        elseif i == 2
            time = '0600';
        elseif i == 3
            time = '1200';
        elseif i == 4
```

```
        time = '1800';
    end
    fctime = '000'; % Forecast time available in 3h steps with 3 digits
    if exist(['simulation/wind_data/wind_',date,'_',time,'.mat'], 'file') ~= 2
        % Forecast
        if mode == 1
            dataname = ['gfs_4_',date,'_',time,'_',fctime,'.grb2'];
            % In case the normal link is not accessible, a second one will be tried
            try
                url = ['https://nomads.ncdc.noaa.gov/data/gfs4/',yyyy,mm,'/',date,'/',dataname];
                filename = ([date,'_',time,'.grb2']);
                cd ('D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data codes\GRIB2\');
                websave(filename,url);
            catch
                url = ['https://www.ncei.noaa.gov/thredds/fileServer/gfs-004-files/',...
                    yyyy,mm,'/',date,'/',dataname];
                filename = ([date,'_',time,'.grb2']);
                cd ('D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data codes\GRIB2\');
                websave(filename,url);
            end

            % Analysis
        elseif mode == 2
            dataname = ['gfsanl_4_',date,'_',time,'_',fctime,'.grb2'];
            % In case the normal link is not accessible, a second one will be tried
            try
                url = ['https://nomads.ncdc.noaa.gov/data/gfsanl/',yyyy,mm,'/',date,'/',dataname];
                filename = ([date,'_',time,'.grb2']);
                cd ('D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data codes\GRIB2\');
                websave(filename,url);
            catch
                url = ['https://www.ncei.noaa.gov/thredds/fileServer/gfs-g4-anl-files/',...
                    yyyy,mm,'/',date,'/',dataname];
                filename = ([date,'_',time,'.grb2']);
                cd ('D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data codes\GRIB2\');
                websave(filename,url);
            end
        end
        cd ..
    else
        cd ..
    end
end
```

B.4. Script process_nc2mat

```
% This script extracts and saves the wind data from the converted nc-files
clc; clear; close all

% Delete grb2-file after extracting and saving wind data? (del = 1: yes,
% del = 2: no)
del = 2;

% Plot wind data? (plotwind = 1: yes, plotwind = 2: no)
plotwind =2;

% Date
year = 2017;
month = 12;
day = 31;
```

```
yyyy = sprintf('%d',year);
if month <10
    mm = sprintf('0%d',month);
else
    mm = sprintf('%d',month);
end
if day <10
    dd = sprintf('0%d',day);
else
    dd = sprintf('%d',day);
end
if month <10
    if day <10
        date = sprintf('%d0%d0%d',year,month,day);
    else
        date = sprintf('%d0%d%d',year,month,day);
    end
else
    if day <10
        date = sprintf('%d%d0%d',year,month,day);
    else
        date = sprintf('%d%d%d',year,month,day);
    end
end

%% Handling nc data
    for i=1:4
        if i == 1
            time = '0000';
        elseif i == 2
            time = '0600';
        elseif i == 3
            time = '1200';
        elseif i == 4
            time = '1800';
        end
        % Get data
        url = ['D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data codes\GRIB2\',date,'_',time,'.nc'];

        % Get wind components
        wind_u = ncread(url,'u-component_of_wind_isobaric');
        wind_u = permute(wind_u, [2 1 3]);
        wind.wind_u{size(wind_u,3)}= wind_u(:,:,1);
        wind_v = ncread(url,'v-component_of_wind_isobaric');
        wind_v = permute(wind_v, [2 1 3]);
        wind.wind_v{size(wind_v,3)}= wind_v(:,:,1);
        wind_vert = ncread(url,'Vertical_velocity_pressure_isobaric');
        wind_vert = permute(wind_vert, [2 1 3]);
        for j = 1:size(wind_u,3)
            wind.wind_u{j}= wind_u(:,:,end-j+1);
            wind.wind_v{j}= wind_v(:,:,end-j+1);
        end
        for j = 1:size(wind_vert,3)
            wind.wind_vert{j}= wind_vert(:,:,end-j+1);
        end
        nday = datenum(year,month,day,str2double(time(1:2)),str2double(time(3:4)),0);
        nlon = ncread(url,'lon');
        nlat = ncread(url,'lat');
```

```
        wind.grid_u{size(wind_u,3)}.time = nday;
        wind.grid_u{size(wind_u,3)}.lon = nlon;
        wind.grid_u{size(wind_u,3)}.lat = nlat;
        wind.grid_v{size(wind_v,3)}.time = nday;
        wind.grid_v{size(wind_v,3)}.lon = nlon;
        wind.grid_v{size(wind_v,3)}.lat = nlat;
        wind.grid_vert{size(wind_vert,3)}.time = nday;
        wind.grid_vert{size(wind_vert,3)}.lon = nlon;
        wind.grid_vert{size(wind_vert,3)}.lat = nlat;
        for j = 1:length(wind.grid_u)
            wind.grid_u{j}.time = nday;
            wind.grid_u{j}.lon = nlon;
            wind.grid_u{j}.lat = nlat;
        end
        for j = 1:length(wind.grid_v)
            wind.grid_v{j}.time = nday;
            wind.grid_v{j}.lon = nlon;
            wind.grid_v{j}.lat = nlat;
        end
        for j = 1:length(wind.grid_vert)
            wind.grid_vert{j}.time = nday;
            wind.grid_vert{j}.lon = nlon;
            wind.grid_vert{j}.lat = nlat;
        end
        wind.p = flipud(ncread(url,'isobaric4'));
        wind.p_vert = flipud(ncread(url,'isobaric3'));
        % Save wind files in easy readable matlab format
        save(['wind_',date,'_',time,'.mat'],'wind');
        % Save at wished location
        source = ['wind_',date,'_',time,'.mat'];
        destination = 'D:\Users\Oriol\Desktop\etseiat\MASTER\TFM\Data
codes\simulation\wind_data';
        movefile(source,destination)
    end
    if plotwind == 1
        figure(1)
        contourf(wind.grid_u{13}.lon,wind.grid_u{13}.lat,wind.wind_u{13},'LineColor','none')
        ylabel({'latitude [º]'});
        xlabel({'longitude [º]'});
        titlename = ['u-component of wind at 25 kPa barometric altitude at ',time(1:2), ':', time(3:4),'
UTC ','[m/s]'];
        title({titlename});
        colorbar
        saveas(figure(1),strcat('Wind plots/u ', time, '.fig'))
        saveas(figure(1),strcat('Wind plots/u ', time,'.png'))
        figure(2)
        contourf(wind.grid_v{13}.lon,wind.grid_v{13}.lat,wind.wind_v{13},'LineColor','none')
        ylabel({'latitude [º]'});
        xlabel({'longitude [º]'});
        titlename = ['v-component of wind at 25 kPa barometric altitude at ',time(1:2), ':', time(3:4),'
UTC ','[m/s]'];
        title({titlename});
        colorbar
        saveas(figure(2),strcat('Wind plots/v ', time,'.fig'))
        saveas(figure(2),strcat('Wind plots/v ', time,'.png'))
    end
    if del == 1
        cd ..;
        delete *.grb2;
```

```
    delete *.grb2.html;
    delete *.grb2.gbx9;
    delete *.grb2.ncx;
end
```

B.5. Script Main_Data_Study

```
% THis script is used to visualize the ascension profile and the trajectory
% of every launch at a certain month and year. Also makes the statistical
% boxplots for steps of 15 min taking into account all these flights.

clc; clear;

%% Data input
k=1;
show_statistics_plots = 1;
show_month_plots = 1;
end_year = 2013;
ini_year = 2013;
end_month = 1;
ini_month = 1;
time_stop = [15 30 45 60];
% Statistics variables inicialization
h_mean =   zeros(12*(end_year-ini_year+1),length(time_stop));
h_median = zeros(12*(end_year-ini_year+1),length(time_stop));
h_min = zeros(12*(end_year-ini_year+1),length(time_stop));
h_max = zeros(12*(end_year-ini_year+1),length(time_stop));
h_stddev = zeros(12*(end_year-ini_year+1),length(time_stop));
h_05percentile = zeros(12*(end_year-ini_year+1),length(time_stop));
h_25percentile = zeros(12*(end_year-ini_year+1),length(time_stop));
h_75percentile = zeros(12*(end_year-ini_year+1),length(time_stop));
h_95percentile = zeros(12*(end_year-ini_year+1),length(time_stop));
var_coefh=zeros(12*(end_year-ini_year+1),length(time_stop));
var_coefh_max=zeros(12*(end_year-ini_year+1),1);
% figure=zeros(12*(end_year-ini_year+1)+200,1);
if show_statistics_plots ==1
    % Create a directory for outputs
    mkdir('Statistics plots');
    mkdir('Month plots');
end
% Year count
for year = ini_year:end_year
    % Month count
    if show_statistics_plots ==1
        % Anual boxplots figure inicialization
        figure('units','normalized','outerposition',[0 0 1 1]);
    end
    for month =ini_month:end_month

        %% File load
        [ DATByyyymm ] = file_load( year, month );
        %% Variables inicialization
        h_time= zeros(max(DATByyyymm.lch_day)*2,length(time_stop));
%         T_time= zeros(max(DATByyyymm.lch_day)*2,length(time_stop));
%         track = zeros(max(DATByyyymm.lch_day)*2,1);
        %% Samples read
        % Launch counter
        i=1;
        % Day count
```

```
    for lch_day=1:31
        % Launch hour count
        for lch_hour = linspace(0,12,2)
            % Data identification by launch
            [DATBsample] = data_identification(DATByyyymm,lch_day, lch_hour);
%               if sum(samples)>0
                %% Ascension interpolation adjustment
                %       % Data identification (x,y)
                %       x = DATBsample.ftr_time(2:length(DATBsample.ftr_time));
                %       y = DATBsample.ftr_alt(2:length(DATBsample.ftr_time));
                %       ceiling = ceil(max(y)/10);
                %       ymin = ceil(min(y)/10);
                %       yy=linspace(ymin,ceiling,(ceiling-ymin)+1)*10;
                %       xx = spline(y,x,yy); xx=xx(xx>=0);
                %       yy = yy((length(yy)-length(xx)+1):length(yy));
                %       plot(x,y,'Color',[0 0 1]); hold on; plot(xx,yy,'Color',[1 0 0]);

                %% Track identification
                % Read the track logs, returns the data in a geoshape object.
                track = geoshape(DATBsample.ftr_LAT', DATBsample.ftr_LON');

                %% Display altitude vs time, T vs altitude
                if show_month_plots ==1
                    figure(k);
                    month_plot( DATBsample);
                    %% Read worldmap
                    figure(k+100);
                    geoplot(DATBsample.ftr_LAT', DATBsample.ftr_LON', 'linewidth',1.5,'Color',
0.8*rand(1,3))
                    geobasemap('landcover');
                    geolimits([min(DATByyyymm.ftr_LAT)
max(DATByyyymm.ftr_LAT)],[min(DATByyyymm.ftr_LON) max(DATByyyymm.ftr_LON)]);
                    % Create title
                    if month <10
                        titlename = sprintf('%d-0%d Map track', year,month);
                    else
                        titlename = sprintf('%d-%d Map track', year,month);
                    end
                    title({titlename});
                    hold on
                end
                %% Data extraction
                h_time(i,:) = time_stops( DATBsample.ftr_alt, DATBsample.ftr_time,time_stop);
                % NAN elimination
                h_time_aux =h_time(i,:);
                h_time_aux(isnan(h_time_aux))=0;
                h_time(i,:)=h_time_aux;
                %               T_time(i,:) = time_stops( DATBsample.ftr_temp,
DATBsample.ftr_time,time_stop);
                i=i+1;
%               end
        end
    end
    if show_month_plots ==1
            % Save figures
            if month <10
                saveas(figure(k), sprintf('Month plots/Altitude & T %d-0%d', year,month), 'jpeg');
                saveas(figure(k+100), sprintf('Month plots/Track map %d-0%d', year,month),
'jpeg');
```

```
            else
                saveas(figure(k), sprintf('Month plots/Altitude & T %d-%d', year,month), 'jpeg');
                saveas(figure(k+100), sprintf('Month plots/Track map %d-0%d', year,month),
'jpeg');
            end
            close(figure(k));
            close(figure(k+100));
        end
        %% Statistics study
        for j=1:length(time_stop)
            h_mean(k,j) = mean(h_time(:,j));
            h_median(k,j) = median(h_time(:,j));
            h_min(k,j) = min(h_time(:,j));
            h_max(k,j) = max(h_time(:,j));
            h_stddev(k,j) = std(h_time(:,j));
            h_05percentile(k,j) = prctile(h_time(:,j),0.05);
            h_25percentile(k,j) = prctile(h_time(:,j),0.25);
            h_75percentile(k,j) = prctile(h_time(:,j),0.75);
            h_95percentile(k,j) = prctile(h_time(:,j),0.95);
        end
        var_coefh(k,:) = h_stddev(k,:)./(abs(h_mean(k,:)));
        var_coefh_max(k) = max(var_coefh(k,:));
        k=k+1;
        if show_statistics_plots ==1
            % Anual boxplots figure inicialization
            figure('units','normalized','outerposition',[0 0 1 1]);
            subplot(3,4,month);
            boxplot_h( h_time./1000, year,month );
            hold on
        end
    end
    if show_statistics_plots ==1
        saveas(figure(k+200), sprintf('Statistics plots/Boxplots %d',year), 'jpeg');
        close(figure(k+200));
    end
end
if show_statistics_plots ==1
    cv_x_axis = ini_year + linspace(1,12*((end_year-ini_year+1)),12*((end_year-
ini_year+1)))./12;
    figure
    plot(cv_x_axis,var_coefh_max)
    % Create xlabel
    xlabel({'Year'});
    % Create ylabel
    ylabel({'Pearson Variation Coefficient [-]'});
    saveas(figure, 'Statistics plots/Variation coeficient', 'jpeg');
    close(figure);
end
```

B.5.1. Function file_load

```
function [ DATByyyymm ] = file_load( year, month )
%file_load Loads a .mat file given month and year
%   Inputs:
%     year          :  Launch year
%     month         :  Launch month
%   Outputs:
%     DATByyyymm    :  Structure with information of launchs registered
```

```
%     during the defined year and month
      if month <10
         filename =
sprintf('GLOBUS_SONDA_UB_RAOB_COMPACT/DATB%d0%d.mat',year,month);
      else
         filename =
sprintf('GLOBUS_SONDA_UB_RAOB_COMPACT/DATB%d%d.mat',year,month);
      end
      DATByyyymm=load(filename, 'DATByyyymm');
      StrName=fieldnames(DATByyyymm);StrName=StrName{1};
      DATByyyymm=DATByyyymm.(StrName);

end
```

B.5.2. Function data_identification

```
function [DATBsample] = data_identification(DATByyyymm,day, hour)
%data_identification Identifies launch data  given day and hour
%   Inputs:
%     DATByyyymm    :  Structure with information of launchs registered
%     year          :  Launch year
%     month         :  Launch month
%   Outputs:
%     DATBsample    :  Structure with information of launch registered
%     during the defined year, month, day and hour

samples = DATByyyymm.lch_day==day & DATByyyymm.lch_hour==hour;
DATBsample.lch_year = DATByyyymm.lch_year(samples);
DATBsample.lch_month = DATByyyymm.lch_month(samples);
DATBsample.lch_day = DATByyyymm.lch_day(samples);
DATBsample.lch_hour = DATByyyymm.lch_hour(samples);
DATBsample.ftr_time = DATByyyymm.ftr_time(samples);          % Temps [s]
DATBsample.ftr_alt = DATByyyymm.ftr_alt(samples);            % Altura [m]
DATBsample.ftr_pres = DATByyyymm.ftr_pres(samples);          % Pressió [hPa/mbar]
DATBsample.ftr_temp = DATByyyymm.ftr_temp(samples);          % Temperatura [ºC]
DATBsample.ftr_hum = DATByyyymm.ftr_hum(samples);            % Humitat relativa [%]
DATBsample.ftr_DP = DATByyyymm.ftr_DP(samples);              % Temperatura del Punt de
Rosada
DATBsample.ftr_WF = DATByyyymm.ftr_WF(samples);              % Velocitat del vent en pla
horitzontal (Wind Force) [m/s o kts]
DATBsample.ftr_WD = DATByyyymm.ftr_WD(samples);              % Direcció del vent [°]
DATBsample.ftr_VEF = DATByyyymm.ftr_VEF(samples);            % Component de vent
horizontal u   [m/s o kts] si + vent oest => est
DATBsample.ftr_VNF = DATByyyymm.ftr_VNF(samples);            % Component de vent
horizontal v  [m/s o kts] si + vent sud => nord
DATBsample.ftr_LAT = DATByyyymm.ftr_LAT(samples);            % Latitud [°]
DATBsample.ftr_LON = DATByyyymm.ftr_LON(samples);            % Longitud [°]
if sum(samples)>0
   [ DATBsample.calc_Vvert ] = vertical_speed_calc( DATBsample.ftr_time, DATBsample.ftr_alt
); % Vertical speed [m/s]
   % Vmod =
(DATBsample.ftr_VEF(2:length(DATBsample.ftr_alt)).^2+DATBsample.ftr_VNF(2:length(DATBs
ample.ftr_alt)).^2+Vvert.^2).^0.5;
   % Vvert2 = (DATBsample.ftr_WF.^2-DATBsample.ftr_VEF.^2-DATBsample.ftr_VNF.^2).^0.5;
   [ DATBsample.calc_Vaccel ] = vertical_accel_calc( DATBsample.ftr_time,
DATBsample.calc_Vvert );
end
end
```

B.5.3. Function vertical_speed_calc

```
function [ Vvert ] = vertical_speed_calc( time, altitude )
%vertical_speed_calc Calculates Vertical Speed from altitude and time data
%   Inputs:
%     time          :  Vector with each timestep
%     altitude      :  Vector with the registered altitude at each timestep
%   Outputs:
%     Vvert         :  Vector with the calculated vertical speed at each
%                      timestep

Vvert = altitude.*0;
Vvert(1)=(altitude(2)-altitude(1))./(time(2)-time(1));
Vvert(2:(length(altitude)-1)) =(altitude(3:(length(altitude)))-altitude(1:(length(altitude)-
2)))./(time(3:length(altitude))-time(1:length(altitude)-2));
Vvert(length(altitude))=(altitude(length(altitude))-altitude(length(altitude)-
1))./(time(length(altitude))-time(length(altitude)-1));
end
```

B.5.4. Function vertical_accel_calc

```
function [ Vaccel ] = vertical_accel_calc( time, Vvert )
%vertical_speed_calc Calculates Vertical Speed from altitude and time data
%   Inputs:
%     time          :  Vector with each timestep
%     Vvert         :  Vector with the calculated vertical speed at each
%                      timestep
%   Outputs:
%     Vaccel        :  Vector with the calculated vertical accel at each
%                      timestep

Vaccel = Vvert.*0;
Vaccel(1)=(Vvert(2)-Vvert(1))./(time(2)-time(1));
Vaccel(2:(length(Vvert)-1)) =(Vvert(3:(length(Vvert)))-Vvert(1:(length(Vvert)-
2)))./(time(3:length(Vvert))-time(1:length(Vvert)-2));
Vaccel(length(Vvert))=(Vvert(length(Vvert))-Vvert(length(Vvert)-1))./(time(length(Vvert))-
time(length(Vvert)-1));
end
```

B.5.5. Function month_plot

```
function month_plot( DATBsample )
%boxplot_h Plots the box plot of the altitudes for each time instant at a
%given month and year
%   Inputs:
%     h           :  Altitude at each time instant for each launch [km]
%     year        :  Launch year
%     month       :  Launch month
%   Outputs:
%     Boxplot of the altitudes for each time instant at a given month and
%     year
year = DATBsample.lch_year(1);
month = DATBsample.lch_month(1);
% day = DATBsample.lch_day(1);

hour = DATBsample.lch_hour(1);
if hour == 0
   color = [0.1*rand(1,2) 1];
else
   color = [1 0.1*rand(1,2)];
end
```

```
subplot(2,1,1);
plot(DATBsample.ftr_time,DATBsample.ftr_alt./1000,'Color',color)
% Create xlabel
xlabel({'time [s]'});
% Create title
if month <10
    titlename = sprintf('%d-0%d Altitude vs time', year,month);
else
    titlename = sprintf('%d-%d Altitude vs time', year,month);
end
title({titlename});
% Create ylabel
ylabel({'altitude [km]'});
% % X-limits of the axes
% xlim([0 7000]);
% % Y-limits of the axes
% ylim([0 30]);
% Create legend
% if day==1 && hour == 12
legend('23 UTC','11 UTC','Location','east');
% end
hold on

subplot(2,1,2);
plot(DATBsample.ftr_alt./1000,DATBsample.ftr_temp,'Color',color)
% Create xlabel
xlabel({'altitude [km]'});
% Create title
if month <10
    titlename = sprintf('%d-0%d Temperature vs altitude', year,month);
else
    titlename = sprintf('%d-%d Temperature vs altitude', year,month);
end
title({titlename});
% Create ylabel
ylabel({'Temperature [ºC]'});
% Create legend
% if  day==1 && hour == 12
legend('23 UTC','11 UTC','Location','east');
% end
hold on
% % X-limits of the axes
% xlim([0 30]);
% % Y-limits of the axes
% ylim([-80 30]);
end
```

B.5.6. Function boxplot_h

```
function boxplot_h( h, year,month )
%boxplot_h Plots the box plot of the altitudes for each time instant at a
%given month and year
%   Inputs:
%     h           :  Altitude at each time instant for each launch [km]
%     year        :  Launch year
%     month       :  Launch month
%   Outputs:
%     Boxplot of the altitudes for each time instant at a given month and
%     year
```

```
boxplot(h)
% Create xlabel
xlabel({'Number of sample (increments of 15 min)'});
% Create title
if month <10
   titlename = sprintf('%d-0%d Boxplot', year,month);
else
   titlename = sprintf('%d-%d Boxplot', year,month);
end
title({titlename});
% Create ylabel
ylabel({'altitude [km]'});


% Create y limit
ylim([0 21])
% saveas(gcf, strcat('Boxplots/', titlename), 'jpeg');

end
```

B.5.7. Function time_stops

```
function [ x_time ] = time_stops( x, time,time_stop )
%time_stops Computes the value of x variable at each of the time stops
%defined
%   Inputs:
%     x           :  Variable array [km, ºC ...]
%     time        :  Time array [s]
%     time_stop   :  Time stops (defined in intervals of 15 min)
%   Outputs:
%     x_time      :  Variable at each time stop

x_time = zeros(length(time_stop),1);
endurance= max(time)/60;
for i = 1:length(time_stop)
   if time_stop(i) < endurance
      x_time(i) = mean(x(time >= time_stop(i)*60-2 & time <= time_stop(i)*60+2));
   end
end
end
```

B.6. Script Main_fit

```
% This script runs the ascension simulation given a date and hour. The
% different temperature models are tested, and in the polytropic case also
% the polytropic index must be introduced. At the end of the script the
% information is saved in a Matlab .mat file to avoid repeating the
% computation.

clear; close all

% save month: 1 to save month data
savemonth = 0;

%% Date of flight
year = 2017;
yyyy = sprintf('%d',year);

%% Temperature Model:
% 1: Balloon Temperature = Ambient Temperature + ideal gas law
```

```
% 2: Polytropic model + ideal gas law
% 3: Polytropic or Isentropic/adiabatic model + ideal gas law
temp_model = [1 2 3];
np_He = 0:0.05:1.6;
np_He(np_He==1)=[];

%% Initialize
for month=[9:12]
    if month <10
        mm = sprintf('0%d',month);
    else
        mm = sprintf('%d',month);
    end
    for day = 14 %1:-1:1
        if day <10
            dd = sprintf('0%d',day);
        else
            dd = sprintf('%d',day);
        end
        date = [yyyy,mm,dd];

        %% Wind data
        if exist(['wind_data/wind_',date,'_','0000','.mat'], 'file') == 2
            wind00 = load(osip(['wind_data/wind_',date,'_','0000','.mat']));
            wind{1} = wind00;
        end
        if exist(['wind_data/wind_',date,'_','0600','.mat'], 'file') == 2
            wind06 = load(osip(['wind_data/wind_',date,'_','0600','.mat']));
            wind{2} = wind06;
        end
        if exist(['wind_data/wind_',date,'_','1200','.mat'], 'file') == 2
            wind12 = load(osip(['wind_data/wind_',date,'_','1200','.mat']));
            wind{3} = wind12;
        end
        if exist(['wind_data/wind_',date,'_','1800','.mat'], 'file') == 2
            wind18 = load(osip(['wind_data/wind_',date,'_','1800','.mat']));
            wind{4} = wind18;
        end

        for hour = linspace(0,12,2)
            %% Load Launch data
            cd ..;
            [ DATByyyymm ] = file_load( year, month );
            [DATBsample] = data_identification(DATByyyymm,day, hour);
            cd simulation;
            if ~isempty(DATBsample.ftr_alt)
                if exist(osip(['wind_data/wind_',date,'_','0000','.mat']), 'file') == 2 ...
                        && exist(osip(['wind_data/wind_',date,'_','0600','.mat']), 'file') == 2 ...
                        && exist(osip(['wind_data/wind_',date,'_','1200','.mat']), 'file') == 2 ...
                        && exist(osip(['wind_data/wind_',date,'_','1800','.mat']), 'file') == 2
                    if hour ==12
                        [simulation(day,1)] = findfit(date, hour, DATBsample, wind, temp_model(1),
np_He(1));
                        [simulation(day,length(np_He)+2)] = findfit(date, hour, DATBsample, wind,
temp_model(3), np_He(1));
                        for i = 2:length(np_He)+1
                            [simulation(day,i)] = findfit(date, hour, DATBsample, wind, temp_model(2),
np_He(i-1));
                        end
```

```
                else
                    [simulation(day+eomday(year,month),1)] = findfit(date, hour, DATBsample, wind,
temp_model(1), np_He(1));
                    [simulation(day+eomday(year,month),length(np_He)+2)] = findfit(date, hour,
DATBsample, wind, temp_model(3), np_He(1));
                    for i = 2:length(np_He)+1
                        [simulation(day+eomday(year,month),i)] = findfit(date, hour, DATBsample,
wind, temp_model(2), np_He(i-1));
                    end
                end

            end
        end
        cd ..; cd Results;
        save(strcat(date,'simulation'), 'simulation')
        cd ..; cd simulation;
    end
  end
end
if savemonth == 1
  cd ..; cd Results;
  save(strcat(date(1:6),'simulation'), 'simulation')
  cd ..; cd simulation;
end
```

B.6.1. Function find_fit

```
function [simulation] = findfit(date, hour, DATBsample, wind,temp_model, np_He)
%findfit Calculates the simulation for each fit which corresponds to
%certain polytropic index for each ascent segment.
%   Inputs:
%     date          :  Date of launch in format yyyymmdd
%     hour          :  Launch hour GMT+1 [h]
%     DATBsample    :  Structure with information of launch registered
%     during the defined year, month, day and hour
%     wind          :  Structure with wind data
%     temp_model    :  Temperature model used in simulation
%       1: Balloon Temperature = Ambient Temperature + ideal gas law
%       2: Polytropic model + ideal gas law
%       3: Polytropic or Isentropic/adiabatic model + ideal gas law
%     np_He         : Segment polytropic index
%   Outputs:
%     simulation    :  Structure containing simulation outputs
%     ECM           :  Structure containing ECM outputs

%% Real launch data
year = str2double(date(1:4));
month = str2double(date(5:6));
day = str2double(date(7:8));
if month ~=1
  dayOfYear = day + sum(eomday(year,1:(month-1)));
else
  dayOfYear = day;
end
real_hour = hour-1;
if real_hour<0
  real_hour = real_hour+24;
end
```

```
%% Parameters
% Ascent
% Starting altitude [m]
z0_a = DATBsample.ftr_alt(1);
% Starting latitude [°]
lat_0_a = DATBsample.ftr_LAT(1);
% Starting longitude [°]
long_0_a = DATBsample.ftr_LON(1);
% Starting velocity [m/s]
vz0_a = 0;
% Starting time [s]
t0 = real_hour*3600;
% Mass of payload [kg]
m_pay = 0.2;
% Mass of balloon [kg]
m_b = 0.35;
% Mass of parachute [kg]
m_p = 0.0; % Approximate value
% Mass of whole structure [kg]
m_s = m_pay+m_b+m_p;
% Initial Helium volume [m^3]
V_0 = 1.1;
%     [T_b0, ~, P_b0, rho_b0, ~] =  AtmosUSSA(z0_a);
[T_b0, ~, P_b0, rho_b0, ~] = atmosNRL(z0_a, lat_0_a, long_0_a, year, dayOfYear, t0);
%     lift_kg = DATBsample.calc_Vaccel(2)*m_s/9.81;
%     syms V_0
%     buoyancy = lift_kg == rho_b0*V_0-(m_s+(4/1000)*P_b0*V_0/(8.314*T_b0));
%     V_0 = double(solve(buoyancy,V_0))
% Burst Radius [m]
hmax = max(DATBsample.ftr_alt);
lat_hmax = DATBsample.ftr_LAT(DATBsample.ftr_alt==max(DATBsample.ftr_alt));
lon_hmax = DATBsample.ftr_LON(DATBsample.ftr_alt==max(DATBsample.ftr_alt));
time_hmax = t0+DATBsample.ftr_time(DATBsample.ftr_alt==max(DATBsample.ftr_alt));
[T_burst, ~, P_burst, ~, ~] = atmosNRL(hmax, lat_hmax, lon_hmax, year, dayOfYear,
time_hmax);
% [T_burst, ~, P_burst, ~, ~] =  AtmosUSSA(hmax);
V_burst = (P_b0*V_0/(8.314*T_b0))*8.314*(T_burst/P_burst);
R_burst = (0.75*V_burst/pi)^(1/3);

%% Plot experiment for live plot
% Altitude
if exist('DATBsample','var')==1
    figure
    % plot(ascend.time_alt,ascend.altitude)
    plot(DATBsample.ftr_time ,DATBsample.ftr_alt)
    if temp_model ==1
        titlename = char(strcat('Isothermal Altitude vs time ',{' '},date, sprintf('%g',hour)));
    elseif  temp_model == 2
        titlename = char(strcat(sprintf('Polytropic %3g Altitude vs time ',np_He),{' '},date,
sprintf('%g',hour)));

    elseif temp_model == 3
        titlename = char(strcat('Adiabatic Altitude vs time ',{' '},date, sprintf('%g',hour)));
    end
    title(titlename)
    hold on
end

tic
```

```
%% Ascent Simulation
[time,state,R,v_wind_rel,t_wind_rel] = ...

ascension(z0_a,vz0_a,year,dayOfYear,t0,temp_model,np_He,m_s,V_0,R_burst,lat_0_a,long_0
_a,wind);
toc

%% Post processing ascent
% Earth radius [m]
R_earth = 6371000;

% Altitude [m]
altitude_ascent = state(:,3);
% Vertical velocity [m/s]
vvel_ascent = diff(altitude_ascent)./diff(time);
time_vvel_ascent = time(1:end-1)+diff(time(:))/2;

% Getting latitudes and longitudes for distances in x and y
% x- and y-coordinates
x_ascent = state(:,1);
y_ascent = state(:,2);
% Transform to lat, long
lat_ascent  = lat_0_a + (y_ascent(1:end-1)./R_earth).*(180/pi);
long_ascent = long_0_a + (x_ascent(1:end-1)/R_earth)*(180/pi)./cos(lat_ascent*pi/180);
% Altitude for latitude and longitude
alt_latlong_ascent = altitude_ascent(1:end-1)+diff(altitude_ascent(1:end))/2;

%% Plot ascent
cd ..; cd Results;
% Altitude
figure(2)
plot(time-time(1),altitude_ascent)
hold on
if exist('DATBsample','var')==1
    plot(DATBsample.ftr_time ,DATBsample.ftr_alt)
    legend('Simulation',['Data (',date,')'])
else
    legend('Simulation')
end
if temp_model ==1
    titlename = char(strcat('Isothermal Altitude vs time ',{' '},date, sprintf('%g',hour)));
elseif  temp_model == 2
    titlename = char(strcat(sprintf('Polytropic %3g Altitude vs time ',np_He),{' '},date,
sprintf('%g',hour)));
elseif temp_model == 3
    titlename = char(strcat('Adiabatic Altitude vs time ',{' '},date, sprintf('%g',hour)));
end
title(titlename)
xlabel('Time [s]')
ylabel('Altitude [m]')
saveas(figure(2),strcat(titlename,'.png'))
% Velocity
figure(3)
plot(time_vvel_ascent,vvel_ascent)
hold on
if exist('DATBsample','var')==1
    plot(DATBsample.ftr_time,DATBsample.calc_Vvert)
    legend('Simulation',['Data (',date,')'])
```

```
else
    legend('Simulation')
end
if temp_model ==1
    titlename = char(strcat('Isothermal Vertical velocity vs time ',{' '},date, sprintf('%g',hour)));
elseif  temp_model == 2
    titlename = char(strcat(sprintf('Polytropic %3g Vertical velocity vs time ',np_He),{' '},date,
sprintf('%g',hour)));

elseif temp_model == 3
    titlename = char(strcat('Adiabatic Vertical velocity vs time ',{' '},date, sprintf('%g',hour)));
end
title(titlename)
xlabel('Time [s]')
ylabel('Vertical velocity [m/s]')
saveas(figure(3),strcat(titlename,'.png'))
% Flight path (trajectory)
figure(4)
if exist('DATBsample','var')==1
    plot3(DATBsample.ftr_LON,DATBsample.ftr_LAT,DATBsample.ftr_alt,'blue');    hold on
end
plot3(long_ascent,lat_ascent,alt_latlong_ascent,'green');   hold on
scatter3(long_ascent(1),lat_ascent(1),alt_latlong_ascent(1),'green','filled',...
    'MarkerEdgeColor',[0 0 0]); hold on
scatter3(long_ascent(end),lat_ascent(end),altitude_ascent(end),'cyan',...
    'filled','MarkerEdgeColor',[0 0 0]); hold on
if exist('DATBsample','var')==1
    scatter3(DATBsample.ftr_LON(end),DATBsample.ftr_LAT(end),DATBsample.ftr_alt(end),...
        'cyan','filled','MarkerEdgeColor',[0 0 0]);
end
xlabel('Longitude [°]'); ylabel('Latitude [°]'); zlabel('Altitude [m]');
if temp_model ==1
    titlename = char(strcat('Isothermal Flight trajectory vs time ',{' '},date, sprintf('%g',hour)));
elseif  temp_model == 2
    titlename = char(strcat(sprintf('Polytropic %3g Flight trajectory vs time ',np_He),{' '},date,
sprintf('%g',hour)));
elseif temp_model == 3
    titlename = char(strcat('Adiabatic Flight trajectory vs time ',{' '},date, sprintf('%g',hour)));
end
title(titlename)
if exist('DATBsample','var')==1
    legend('Data (Ascension)','Simulation (Ascension)', 'Launch site',...
        'Balloon burst (Simulation)','Balloon burst (Data)')
else
    legend('Simulation (Ascension)','Launch site',...
        'Balloon burst (Simulation)')
end
saveas(figure(4),strcat(titlename,'.png'))
% Relative wind velocity
figure(5)
plot(t_wind_rel,v_wind_rel(:,1),t_wind_rel,v_wind_rel(:,2),...
    t_wind_rel,v_wind_rel(:,3))
xlabel('Time [s]')
ylabel('Relative velocity [m/s]')
legend('Relative wind in x-direction','Relative wind in y-direction',...
    'Relative wind in z-direction')
if temp_model ==1
    titlename = char(strcat('Isothermal Relative 3D velocity between wind and balloon',{' '},date,
sprintf('%g',hour)));
```

```
elseif  temp_model == 2
   titlename = char(strcat(sprintf('Polytropic %3g Relative 3D velocity between wind and
balloon',np_He),{' '},date, sprintf('%g',hour)));

elseif temp_model == 3
   titlename = char(strcat('Adiabatic Relative 3D velocity between wind and balloon',{' '},date,
sprintf('%g',hour)));
end
title(titlename)
saveas(figure(5),strcat(titlename,'.png'));
close all;
cd ..; cd simulation;

%% Outputs
simulation.time=time;
simulation.state=state;
simulation.R=R;
simulation.v_wind=v_wind_rel;
simulation.t_wind=t_wind_rel;

end
```

B.6.2. Function ascension

```
function [t,state,Radius,v_wind_rel,t_wind_rel] = ascension(varargin)
% This function simulates the ascension of a helium-filled sounding
% balloon
%
% Input (in form of varargin):
%   - z0: Start height [m]
%   - vz0: Vertical velocity at start [m/s]
%   - t0: Starting time [s]
%   - option: Temperature model
%   - m_pay: Payload mass + balloon and parachute mass [kg]
%   - V_0: Starting helium volume [m³]
%   - R_burst: Radius at which balloon bursts [m]
%   - lat_0: Starting latitude [°]
%   - long_0: Starting longitude [°]
%   - wind_dat: Wind data [m/s]
%
% Output:
%   - time: Time vector of simulation [s]
%   - state: Position and velocity in x-,y- and z-direction for the
%            whole ascension [m]/[m/s]
%   - R: Final radius [m]
%   - v_wind_rel: Relative wind velocity vector [m/s]
%   - t_wind_rel: Relative wind velocity time vector [s]
%
% Manuel Schubert, UPC
% 2018

%% Data and initialisation
z0 = varargin{1};          % Starting altitude [m]
vz0 = varargin{2};         % Starting velocity [m/s]
if vz0 == 0, vz0 = 1E-20; end
year = varargin{3};
dayOfYear = varargin{4};
t0 = varargin{5};          % Starting time [s]
option = varargin{6};      % Temperature Model
```

```
np_He = varargin{7};        % Polytropic index
m_pay = varargin{8};        % Mass of payload [kg]
V_0 = varargin{9};          % Initial Helium volume [m^3]
R_burst = varargin{10};     % Burst Radius [m]
lat_0 = varargin{11};       % Starting latitude [°]
long_0 = varargin{12};      % Starting longitude [°]
wind_dat = varargin{13};    % Wind data [m/s]

% Effective radius [m]
R0 = (V_0*(3/4)/pi).^(1/3);

% Earth radius [m]
R_earth = 6371;

% Initial ambient air parameters
% [T_a0, ~, p_a0, ~, ~] =  AtmosUSSA(z0);
[T_a0, ~, p_a0, ~, ~] = atmosNRL(z0, lat_0, long_0, year, dayOfYear, t0);

% Initial balloon temperature equals initial ambient temperature [K]
T_b0 = T_a0;
T_bt = T_b0;

% Thermodynamical properties
R_universal = 8.314;        % Universal gas constant [kgm^2/s^2molK]
rho_He = 0.1685;            % Density of Helium [kg/m^3]
gamma_He = 1.666667;    % Heat capacity ratio He
Cp_He = 5190;               % Specific isobar heat capacity [J/KgK]
kappa_He = 0.149;           % Thermal conductivity [W/m^2K]

% Helium mass [kg]
n_He = p_a0*V_0/(R_universal*mean(T_a0));
m_He = 4e-3*n_He;

% Prepare wind data
% Pressure steps for wind data
p_wind = wind_dat{1,1}.wind.p;
p_wind_vert = wind_dat{1,1}.wind.p_vert;

% Grid
wind.grid_v = wind_dat{1,1}.wind.grid_v;
wind.grid_u = wind_dat{1,1}.wind.grid_u;

% Time steps for wind data
t_wind_steps = [0,6,12,18]*3600;

% Simulation
% Set initial radius
R = R0;

% Time Span
tspan = t0:1:t0+10000;
% Options for solver
options = odeset('Events',@ReachingTop,'RelTol',1e-10,'AbsTol',1e-3,...
    'OutputFcn',@odeplot,'OutputSel',3);
% options = odeset('Events',@ReachingTop,'RelTol',1e-10,'AbsTol',1e-3);
% Initial input
x0 = 0; vx0 = 1E-20; y0 = 0; vy0 = 1E-20;
state0 = [x0;y0;z0;vx0;vy0;vz0];
```

```
% Count variables for event
ii = 1; yy = 0; xx = 0; zz = 0;

% Solving of the differential equation
[t,state] = ode45(@rise,tspan,state0,options);

    %% Differential equation of ascension
    function [X] = rise(t,state)
    % This function solves the differential equation for the ascension
    % for every time step

    % Position from input [m]
    x = state(1);
    y = state(2);
    z = state(3);
    % Velocity from input [m/s]
    vx = state(4);
    vy = state(5);
    vz_star = state(6);


    %% Wind velocity [m/s]
        % Actual time
        t_actual = t0+t;
        t_first = max(t_wind_steps((t_wind_steps<=t_actual)==1));
        t_second = min(t_wind_steps((t_wind_steps>=t_actual)==1));

        % Actual position in lat/long
        if size(yy)>1
        lat  = lat_0 + (yy(1:end)/R_earth)*(180/pi);
        long = long_0 + (xx(1:end)/R_earth)*(180/pi)./cos(lat*pi/180);
        else
            lat = lat_0; long = long_0;
        end

        if long < 0
            long = long+360;
        end

        % Get atmospheric properties from USSA and gravity
        [g,~] = GeopotentialModel(z);
%       [T_a, ~, p_a, rho_a, mu_a] =  AtmosUSSA(z);
        [T_a, ~, p_a, rho_a, mu_a] = atmosNRL(z, lat, long, year, dayOfYear, t_actual);
        % Find corresponding values for actual pressure, lat and long
            % Pressure
            if p_wind(1)<p_a
                pos1_a = 1;
            else
                pos1_a = find(p_wind==min(p_wind((p_a<=p_wind)==1)));
            end
            if p_a < min(p_wind)
                pos1_b = pos1_a;
            else
                pos1_b = find(p_wind==max(p_wind((p_a>=p_wind)==1)));
            end
                if abs(p_wind(pos1_a)-p_a)<abs(p_wind(pos1_b)-p_a)
                    pos1 = pos1_a;
                else
```

```
            pos1 = pos1_b;
        end

    % Latitude
    pos2_a = find(wind.grid_v{pos1}.lat==...
        min(wind.grid_v{pos1}.lat((lat<=...
        wind.grid_v{pos1}.lat)==1)));
    pos2_b = find(wind.grid_v{pos1}.lat==...
        max(wind.grid_v{pos1}.lat((lat>=...
        wind.grid_v{pos1}.lat)==1)));

    % Longitude
    pos3_a = find(wind.grid_v{pos1}.lon==...
        min(wind.grid_v{pos1}.lon((long<=...
        wind.grid_v{pos1}.lon)==1)));
    if long <=360 && long>= 359.5
        pos3_b = find(wind.grid_v{pos1}.lon==...
            min(wind.grid_v{pos1}.lon((long>=...
            wind.grid_v{pos1}.lon)==1)));
    else
        pos3_b = find(wind.grid_v{pos1}.lon==...
            min(wind.grid_v{pos1}.lon((long<=...
            wind.grid_v{pos1}.lon)==1)));
    end

    % Positions for vertical velocity pressure
    if p_wind_vert(1)<p_a
        pos4_a = 1;
    else
        pos4_a = find(p_wind_vert==min(p_wind_vert((p_a<=p_wind_vert)==1)));
    end
    if p_a < min(p_wind_vert)
        pos4_b = pos4_a;
    else
        pos4_b = find(p_wind_vert==max(p_wind_vert((p_a>=p_wind_vert)==1)));
    end

% Interpolation for the time
    for jj=1:2
        % Choosing wind corresponding to time
        if jj == 1
            if t_first == 0
                wind_x = wind_dat{1,1}.wind.wind_u;
                wind_y = wind_dat{1,1}.wind.wind_v;
                wind_z = wind_dat{1,1}.wind.wind_vert;
            elseif t_first == 6*3600
                wind_x = wind_dat{1,2}.wind.wind_u;
                wind_y = wind_dat{1,2}.wind.wind_v;
                wind_z = wind_dat{1,2}.wind.wind_vert;
            elseif t_first == 12*3600
                wind_x = wind_dat{1,3}.wind.wind_u;
                wind_y = wind_dat{1,3}.wind.wind_v;
                wind_z = wind_dat{1,3}.wind.wind_vert;
            elseif t_first == 18*3600
                wind_x = wind_dat{1,4}.wind.wind_u;
                wind_y = wind_dat{1,4}.wind.wind_v;
                wind_z = wind_dat{1,4}.wind.wind_vert;
            end
        elseif jj == 2
```

```
        if t_second == 0
            wind_x = wind_dat{1,1}.wind.wind_u;
            wind_y = wind_dat{1,1}.wind.wind_v;
            wind_z = wind_dat{1,1}.wind.wind_vert;
        elseif t_second == 6*3600
            wind_x = wind_dat{1,2}.wind.wind_u;
            wind_y = wind_dat{1,2}.wind.wind_v;
            wind_z = wind_dat{1,2}.wind.wind_vert;
        elseif t_second == 12*3600
            wind_x = wind_dat{1,3}.wind.wind_u;
            wind_y = wind_dat{1,3}.wind.wind_v;
            wind_z = wind_dat{1,3}.wind.wind_vert;
        elseif t_second == 18*3600
            wind_x = wind_dat{1,4}.wind.wind_u;
            wind_y = wind_dat{1,4}.wind.wind_v;
            wind_z = wind_dat{1,4}.wind.wind_vert;
        end
    end
% Corresponding wind velocities to altitude (interpolation between
% altitude, longitude and latitude (8 points))
    % X-direction (8 nearest points around actual point)
    v_wind_x_000 = wind_x{pos1_a}(pos2_a,pos3_a);
    v_wind_x_200 = wind_x{pos1_b}(pos2_a,pos3_a);
    v_wind_x_002 = wind_x{pos1_a}(pos2_a,pos3_b);
    v_wind_x_202 = wind_x{pos1_b}(pos2_a,pos3_b);
    v_wind_x_020 = wind_x{pos1_a}(pos2_b,pos3_a);
    v_wind_x_220 = wind_x{pos1_b}(pos2_b,pos3_a);
    v_wind_x_022 = wind_x{pos1_a}(pos2_b,pos3_b);
    v_wind_x_222 = wind_x{pos1_b}(pos2_b,pos3_b);
    % X-direction (Alitude interpolated)
    if pos1_a == pos1_b
        v_wind_x_100 = v_wind_x_000;
        v_wind_x_102 = v_wind_x_002;
        v_wind_x_120 = v_wind_x_020;
        v_wind_x_122 = v_wind_x_022;
    else
        v_wind_x_100 = v_wind_x_000+((v_wind_x_200-v_wind_x_000)...
        /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
        v_wind_x_102 = v_wind_x_002+((v_wind_x_202-v_wind_x_002)...
        /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
        v_wind_x_120 = v_wind_x_020+((v_wind_x_220-v_wind_x_020)...
        /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
        v_wind_x_122 = v_wind_x_022+((v_wind_x_222-v_wind_x_022)...
        /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
    end
    % X-direction (Longitude interpolated)
    if pos3_a == pos3_b
        v_wind_x_101 = v_wind_x_100;
        v_wind_x_121 = v_wind_x_120;
    else
        v_wind_x_101 = v_wind_x_100+((v_wind_x_102-v_wind_x_100)...
            /(wind.grid_u{pos1}.lon(pos3_b)-...
            wind.grid_u{pos1}.lon(pos3_a)))*...
            (long-wind.grid_u{pos1}.lon(pos3_a));
        v_wind_x_121 = v_wind_x_120+((v_wind_x_122-v_wind_x_120)...
            /(wind.grid_u{pos1}.lon(pos3_b)-...
            wind.grid_u{pos1}.lon(pos3_a)))*...
            (long-wind.grid_u{pos1}.lon(pos3_a));
```

```
end
% X-direction (Latitude interpolated)
if pos2_a == pos2_b
   v_wind_x(jj) = v_wind_x_101;
else
   v_wind_x(jj) = v_wind_x_101+((v_wind_x_121-v_wind_x_101)...
      /(wind.grid_u{pos1}.lat(pos2_b)-...
      wind.grid_u{pos1}.lat(pos2_a)))*...
      (lat-wind.grid_u{pos1}.lat(pos2_a));
end

% Y-direction (8 nearest points around actual point)
v_wind_y_000 = wind_y{pos1_a}(pos2_a,pos3_a);
v_wind_y_200 = wind_y{pos1_b}(pos2_a,pos3_a);
v_wind_y_002 = wind_y{pos1_a}(pos2_a,pos3_b);
v_wind_y_202 = wind_y{pos1_b}(pos2_a,pos3_b);
v_wind_y_020 = wind_y{pos1_a}(pos2_b,pos3_a);
v_wind_y_220 = wind_y{pos1_b}(pos2_b,pos3_a);
v_wind_y_022 = wind_y{pos1_a}(pos2_b,pos3_b);
v_wind_y_222 = wind_y{pos1_b}(pos2_b,pos3_b);
% Y-direction (Alitude interpolated)
if pos1_a == pos1_b
   v_wind_y_100 = v_wind_y_000;
   v_wind_y_102 = v_wind_y_002;
   v_wind_y_120 = v_wind_y_020;
   v_wind_y_122 = v_wind_y_022;
else
   v_wind_y_100 = v_wind_y_000+((v_wind_y_200-v_wind_y_000)...
   /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
   v_wind_y_102 = v_wind_y_002+((v_wind_y_202-v_wind_y_002)...
   /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
   v_wind_y_120 = v_wind_y_020+((v_wind_y_220-v_wind_y_020)...
   /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
   v_wind_y_122 = v_wind_y_022+((v_wind_y_222-v_wind_y_022)...
   /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
end

% Y-direction (Longitude interpolated)
if pos3_a == pos3_b
   v_wind_y_101 = v_wind_y_100;
   v_wind_y_121 = v_wind_y_120;
else
   v_wind_y_101 = v_wind_y_100+((v_wind_y_102-v_wind_y_100)...
      /(wind.grid_u{pos1}.lon(pos3_b)-...
      wind.grid_u{pos1}.lon(pos3_a)))*...
      (long-wind.grid_u{pos1}.lon(pos3_a));
   v_wind_y_121 = v_wind_y_120+((v_wind_y_122-v_wind_y_120)...
      /(wind.grid_u{pos1}.lon(pos3_b)-...
      wind.grid_u{pos1}.lon(pos3_a)))*...
      (long-wind.grid_u{pos1}.lon(pos3_a));
end
% Y-direction (Latitude interpolated)
if pos2_a == pos2_b
   v_wind_y(jj) = v_wind_y_101;
else
   v_wind_y(jj) = v_wind_y_101+((v_wind_y_121-v_wind_y_101)...
      /(wind.grid_u{pos1}.lat(pos2_b)-...
      wind.grid_u{pos1}.lat(pos2_a)))*...
      (lat-wind.grid_u{pos1}.lat(pos2_a));
```

```
        end

        % Z-direction (8 nearest points around actual point)
        v_wind_z_000 = wind_z{pos4_a}(pos2_a,pos3_a);
        v_wind_z_200 = wind_z{pos4_b}(pos2_a,pos3_a);
        v_wind_z_002 = wind_z{pos4_a}(pos2_a,pos3_b);
        v_wind_z_202 = wind_z{pos4_b}(pos2_a,pos3_b);
        v_wind_z_020 = wind_z{pos4_a}(pos2_b,pos3_a);
        v_wind_z_220 = wind_z{pos4_b}(pos2_b,pos3_a);
        v_wind_z_022 = wind_z{pos4_a}(pos2_b,pos3_b);
        v_wind_z_222 = wind_z{pos4_b}(pos2_b,pos3_b);
        % Z-direction (Alitude interpolated)
        if pos4_a == pos4_b
            v_wind_z_100 = v_wind_z_000;
            v_wind_z_102 = v_wind_z_002;
            v_wind_z_120 = v_wind_z_020;
            v_wind_z_122 = v_wind_z_022;
        else
            v_wind_z_100 = v_wind_z_000+((v_wind_z_200-v_wind_z_000)...
            /(p_wind_vert(pos4_b)-p_wind_vert(pos4_a)))*(p_a-p_wind_vert(pos4_a));
            v_wind_z_102 = v_wind_z_002+((v_wind_z_202-v_wind_z_002)...
            /(p_wind_vert(pos4_b)-p_wind_vert(pos4_a)))*(p_a-p_wind_vert(pos4_a));
            v_wind_z_120 = v_wind_z_020+((v_wind_z_220-v_wind_z_020)...
            /(p_wind_vert(pos4_b)-p_wind_vert(pos4_a)))*(p_a-p_wind_vert(pos4_a));
            v_wind_z_122 = v_wind_z_022+((v_wind_z_222-v_wind_z_022)...
            /(p_wind_vert(pos4_b)-p_wind_vert(pos4_a)))*(p_a-p_wind_vert(pos4_a));
        end
        % Z-direction (Longitude interpolated)
        if pos3_a == pos3_b
            v_wind_z_101 = v_wind_z_100;
            v_wind_z_121 = v_wind_z_120;
        else
            v_wind_z_101 = v_wind_z_100+((v_wind_z_102-v_wind_z_100)...
                /(wind.grid_u{pos1}.lon(pos3_b)-...
                wind.grid_u{pos1}.lon(pos3_a)))*...
                (long-wind.grid_u{pos1}.lon(pos3_a));
            v_wind_z_121 = v_wind_z_120+((v_wind_z_122-v_wind_z_120)...
                /(wind.grid_u{pos1}.lon(pos3_b)-...
                wind.grid_u{pos1}.lon(pos3_a)))*...
                (long-wind.grid_u{pos1}.lon(pos3_a));
        end
        % Z-direction (Latitude interpolated)
        if pos2_a == pos2_b
            v_wind_z(jj) = v_wind_z_101;
        else
            v_wind_z(jj) = v_wind_z_101+((v_wind_z_121-v_wind_z_101)...
                /(wind.grid_u{pos1}.lat(pos2_b)-...
                wind.grid_u{pos1}.lat(pos2_a)))*...
                (lat-wind.grid_u{pos1}.lat(pos2_a));
        end
        end
if v_wind_x(1) == v_wind_x(2)
    v_wind_x = v_wind_x(1);
else
   v_wind_x = v_wind_x(1)+((v_wind_x(2)-v_wind_x(1))...
        /(t_second-t_first))*(t_actual-t_first);
end
if v_wind_y(1) == v_wind_y(2)
```

```
    v_wind_y = v_wind_y(1);
else
  v_wind_y = v_wind_y(1)+((v_wind_y(2)-v_wind_y(1))...
        /(t_second-t_first))*(t_second-t_actual);
end
if v_wind_z(1) == v_wind_z(2)
    v_wind_z = v_wind_z(1);
else
   v_wind_z = v_wind_z(1)+((v_wind_z(2)-v_wind_z(1))...
        /(t_second-t_first))*(t_second-t_actual);
end

%% Balloon Temperature and Radius depending on model
if option == 1
    T_b = T_a;
    V = n_He*R_universal*T_b./p_a;
    R = (V*(3/4)/pi).^(1/3);
elseif option == 2
   % Polytropic
   T_b = power((p_a/p_a0),((np_He-1)/np_He))*T_bt;
   V = n_He*R_universal*T_b./p_a;
   R = (V*(3/4)/pi).^(1/3);
elseif option ==3
   % Isentropic / Adiabatic
   T_b = power((p_a/p_a0),((gamma_He-1)/gamma_He))*T_bt;
   V = n_He*R_universal*T_b./p_a;
   R = (V*(3/4)/pi).^(1/3);
end

%% Calculate total mass [kg]
V = (R^3)*(4/3)*pi;
% Effective area [m²]
A = pi*R^2;

% Virtual mass [kg]
m_a = rho_a * V;
% Total balloon mass [kg]
m_tot = m_He+m_pay;

%% Get Cd
Re = (rho_a*R*vz_star)/mu_a;

% Drag coefficient [1]
Cd = 0.04808*(log(Re))^2-1.406*log(Re)+10.49;
Cd = real(Cd);

%% Calculate acceleration forces
F_Drag_x = 0.5*Cd*A*rho_a*state(4)*abs(state(4)); % Drag Force x
F_Drag_y = 0.5*Cd*A*rho_a*state(5)*abs(state(5)); % Drag Force y
F_Drag_z = 0.5*Cd*A*rho_a*state(6)*abs(state(6)); % Drag Force z
F_Lift = (m_a - m_tot)*g;          % Lift Force

% Wind forces
F_wind_x = 0.5*rho_a*v_wind_x*abs(v_wind_x)*A;
F_wind_y = 0.5*rho_a*v_wind_y*abs(v_wind_y)*A;
F_wind_z = 0.5*rho_a*v_wind_z*abs(v_wind_z)*A;

% Acceleration [m/s²]
dvx = (F_wind_x-F_Drag_x)/m_tot;
```

```
    dvy = (F_wind_y-F_Drag_y)/m_tot;
    dvz = (F_Lift-F_Drag_z+F_wind_z)/m_tot;

    %% Define output (velocity [m/s] and acceleration [m/s²])
    X = [state(4);state(5);state(6);dvx;dvy;dvz]; X = double(X);

    % Relative wind speed
    v_wind_rel(ii,1:3) = [state(4)-v_wind_x;state(5)-v_wind_y;...
        state(6)-v_wind_z];
    t_wind_rel(ii) = t;

    % Parameters of step n stored for step n+1
    p_a0 = p_a;
    T_bt = T_b;

    end

    %% Event Function to stop ode45
    function [value,isterminal,direction] = ReachingTop(t,state)
    % This function serves as event to stop the differential equation
    % solver and therefore the simulation
    xx(ii) = state(1); yy(ii) = state(2); zz(ii) = state(3); Radius(ii) = R;

    % End the simulation, when balloon reaches highest point on the
    % trajectory (when it "stops" to rise) or the balloon bursts
    if ii>20
        % End Condition
        value = double(mean(zz(ii-20:ii)-zz(ii-20)) > 5E-1 & R_burst >= R);
    else
        value = 1;
    end
    isterminal = 1;     % End calculation after condition is met
    direction = 0;      % Local Minimum/Maximum
    ii = ii+1;
    end

end
```

B.6.3. Function atmos_NRL

```
function [T, a, P, rho, mu] = atmosNRL(h, lat, lon, year, dayOfYear, UTseconds)
%%atmosNRL NRLMSISE-00 Atmospheric model function.
%   Inputs:
%     h           :  Altitude
%     lat         :  Latitude
%     lon         :  Longitude
%     year        :  Launch year
%     dayOfYear   :  Day of year (1-365/366)
%     UTseconds   :  Time of day [s]
%   Outputs:
%     DATBsample  :  Structure with information of launch registered
%     during the defined year, month, day and hour
%
% Inputs:
%   - h: Altitude [m]
%
% Outputs:
%   - T: Temperature [K]
%   - a: Speed of sound [m/s]
```

```
%   - P: Pressure [Pa]
%   - rho: Density [kg/m3]
%   - mu: Dynamic viscosity [Ns/m^2]
% Return the temperature and density
[T, rho] = atmosnrlmsise00( h, lat, lon, year, dayOfYear, UTseconds);
T = T(:,2);
% Return the speed of sound
a=sqrt(1.4*287*T);
rho = rho(:,6);
% Return the pressure
P=rho.*(287*T);
% Return the dynamic viscosity
mu=(1.458e-06).*(T^1.5)/(T+110.4);
```

B.6.4. Function GeopotentialModel

```
function [gx,gz] = GeopotentialModel(h,varargin)
% This function calculates the gravity acceleration in x- and z- direction
% using the geopotential gravity model
%
% function [gx,gz] = GeopotentialModel(h,varargin)
%
% Inputs:
%   - h: Altitude [m]
%   - mu: Standard gravitational parameter [m³/s²]
%   - R: Radius of the planet [m]
%
% Outputs:
%   - gx: Gravity acceleration [m/s²]
%   - gz: Gravity acceleration [m/s²]
%
% Reference:
%   [1]: Arnau Miró, Manel Soria, Suborbital Flight Dynamics, 2016

% Get inputs from varargin
if isempty(varargin)
    mu = 3.986004418E14;
    R = 6371000.785;
else
    mu = varargin{1};
    R = varargin{2};
end

% Radius of the orbit [m]
r = h+R;

% Calculation of the gravitational acceleration components [m/s²]
% x component
gx = mu/power(r,2);
% z component
gz = 0;

end
```

B.7. Script MAIN_Plot_day

```
% This script plots altitude, vertical velocity, trajectory, and relative
% wind for a given day and several values of polytropic index

clear;close all;clc;
```

```
%% Input data
% Date
year = 2017; month = 2; day =14; hour = 0;

%% Experimental Data load
[ DATByyyymm ] = file_load( year, month );
[DATBsample] = data_identification(DATByyyymm,day, hour);

%% Simulation Data load
% Date
yyyy = sprintf('%d',year);
if month <10
    mm = sprintf('0%d',month);
else
    mm = sprintf('%d',month);
end
if day <10
    dd = sprintf('0%d',day);
else
    dd = sprintf('%d',day);
end
date = [yyyy,mm,dd];
cd Results;
load(strcat(date,'simulation'), 'simulation')
cd ..;
% Polytropic index
np_He = 0:0.05:1.6;
np_He(np_He==1)=[];
np_He = [1,np_He,1.67];
if hour == 12
    simulation = simulation(day,:);
elseif hour == 0
    simulation = simulation(day+eomday(year,month),:);
end
cd Results;

%% Altitude plot
for j = 1:length(np_He)
    np_sel = np_He(j);
    % Simulation data for selected polytropic index
    sim = simulation(np_He==np_sel);
    time = sim.time-sim.time(1);                % time [s]
    altitude_ascent = sim.state(:,3);           % Altitude [m]
    figure(2)
    plot1(j) = plot(time,altitude_ascent,'Color', [0, rand(1,2)]);
    % Create multiple lines using matrix input to plot
    DisplayName = sprintf('n = %3g',np_He(j));
    set(plot1(j),'DisplayName',DisplayName);
    hold on
end
if exist('DATBsample','var')==1
    plot1(j+1) = plot(DATBsample.ftr_time ,DATBsample.ftr_alt,'LineWidth',2,'Color', 'r');
    set(plot1(j+1),'DisplayName',['Data (',date,')']);
else
end
% Create legend
legend('show');
set(legend,'NumColumns',4,'Location','southeast');
```

```
titlename = char(strcat(date, sprintf('%g',hour), ' Altitude vs time'));
title(titlename)
xlabel('Time [s]')
ylabel('Altitude [m]')
% Enlarge figure to full screen.
set(gcf, 'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);
saveas(figure(2),strcat(titlename,'.fig'))
cd ..;

%% Change of slope point calculation
limit = floor(length(DATBsample.calc_Vvert)/100);
Vvert_av100 = zeros(1,limit*100);
time_av100 = DATBsample.ftr_time(1:limit*100);
for j = 1:limit
    step = j*100;
    Vvert_av100(step-99:step) = mean(DATBsample.calc_Vvert(step-99:step));
end
figure; plot(time_av100,Vvert_av100,'LineWidth',2,'Color', 'r')
legend(['Simulation',' n = ', sprintf('%g',np_sel)],['Data (',date,')'])

diference = abs(Vvert_av100(2:end)-Vvert_av100(1:end-1))';
time_change = time_av100(diference==max(diference));

%% Fit calculations
% 1st segment
ECM_1st = zeros(length(np_He),1);
for i = 1:length(np_He)
    np_sel = np_He(i);
    sim = simulation(np_He==np_sel);            % Simulation data for selected polytropic index
    time = sim.time-sim.time(1);                % time [s]
    altitude_ascent = sim.state(:,3);           % Altitude [m]
    time_max = min(time_change,max(time));
    error_q_alt=zeros(length(DATBsample.ftr_alt(DATBsample.ftr_time<time_max)),1);

    for j = 1:length(error_q_alt)
        error_q_alt(j) = (DATBsample.ftr_alt(j) - altitude_ascent(time==DATBsample.ftr_time(j)))^2;
    end
    if time_max == time_change
        ECM_1st(i) = sum(error_q_alt)/size(error_q_alt,2);
    else
        ECM_1st(i) = max(ECM_1st)+1;
    end
end
np_He_1st = np_He(ECM_1st==min(ECM_1st));

% 2nd segment
ECM_2nd = zeros(length(np_He),1);
for i = 1:length(np_He)
    np_sel = np_He(i);
    sim = simulation(np_He==np_sel);            % Simulation data for selected polytropic index
    time = sim.time-sim.time(1);                % time [s]
    vvel_ascent = sim.state(:,6);               % Vertical speed [m/s]
    time_min = time_change;
    time_max = DATBsample.ftr_time(end);
    error_q_Vvert=zeros(length(DATBsample.ftr_alt(DATBsample.ftr_time>=time_min)),1);

    time_change_index = find(DATBsample.ftr_time==time_min);
    for j = 1:length(error_q_Vvert)
        if and(max(time)>time_min,max(time)>=time_max)
```

```
        error_q_Vvert(j) = (DATBsample.calc_Vvert(time_change_index+j-1) - ...
vvel_ascent(time==DATBsample.ftr_time(DATBsample.ftr_time==DATBsample.ftr_time(time_ch
ange_index+j-1))))^2;
      else
        error_q_Vvert(j) = 1000;
      end
   end
   if and(max(time)>time_min,max(time)>=time_max)
      ECM_2nd(i) = sum(error_q_Vvert)/size(error_q_Vvert,2);
   else
      ECM_2nd(i) = max(ECM_2nd)+1;
   end
end
np_He_2nd = np_He(ECM_2nd==min(ECM_2nd));

%% Outputs
fprintf('Slope change point at t = %1.0f s \n', time_change)
fprintf('Segment 1 polytropic index = %1.2f \n', np_He_1st)
fprintf('Segment 2 initial polytropic index = %1.2f \n', np_He_2nd)
```

B.8. Script Main_Fit_Day

```
% This script is used to define the best fit using the initial estimations
% as well as the point of change of slopefound in script Main_Plot_Day.

clear; close all

% save month: 1 to save month data
savemonth = 0;
save = 0;

%% Main_Plot_Day outputs
slope_point = 1730;        % Slope point time [s]
np_He_1st = 0.6;           % Polytropic index of semgent 1
np_He_2nd = 0.95;          % Polytropic index of semgent 2

%% Date of flight
year = 2017;
month=4;
day = 14;
hour =12;
yyyy = sprintf('%d',year);
if month <10
   mm = sprintf('0%d',month);
else
   mm = sprintf('%d',month);
end
if day <10
   dd = sprintf('0%d',day);
else
   dd = sprintf('%d',day);
end
date = [yyyy,mm,dd];

%% Temperature Model:
% 1: Balloon Temperature = Ambient Temperature + ideal gas law
% 2: Polytropic model + ideal gas law
% 3: Polytropic or Isentropic/adiabatic model + ideal gas law
```

```
temp_model = 2;

%% Wind data
if exist(['wind_data/wind_',date,'_','0000','.mat'], 'file') == 2
    wind00 = load(['wind_data/wind_',date,'_','0000','.mat']);
    wind{1} = wind00;
end
if exist(['wind_data/wind_',date,'_','0600','.mat'], 'file') == 2
    wind06 = load(['wind_data/wind_',date,'_','0600','.mat']);
    wind{2} = wind06;
end
if exist(['wind_data/wind_',date,'_','1200','.mat'], 'file') == 2
    wind12 = load(['wind_data/wind_',date,'_','1200','.mat']);
    wind{3} = wind12;
end
if exist(['wind_data/wind_',date,'_','1800','.mat'], 'file') == 2
    wind18 = load(['wind_data/wind_',date,'_','1800','.mat']);
    wind{4} = wind18;
end

    %% Load Launch data
    cd ..; cd ..;
    [ DATByyyymm ] = file_load( year, month );
    [DATBsample] = data_identification(DATByyyymm,day, hour);
    cd simulation;
    if ~isempty(DATBsample.ftr_alt)
        if exist(['wind_data/wind_',date,'_','0000','.mat'], 'file') == 2 ...
                && exist(['wind_data/wind_',date,'_','0600','.mat'], 'file') == 2 ...
                && exist(['wind_data/wind_',date,'_','1200','.mat'], 'file') == 2 ...
                && exist(['wind_data/wind_',date,'_','1800','.mat'], 'file') == 2

            %% 1st segment Simulation
            % Starting time [h]
            real_hour = hour-1;
            if real_hour<0
                real_hour = real_hour+24;
            end
            % Starting time [s]
            ini.t0 = real_hour*3600;
            % Simulation time deviation [s]
            ini.sim_time = 0;
            % Ascent parameters
            % Earth radius [m]
            R_earth = 6371000;
            % Polytropic index
            np_He1 = np_He_1st;
            % Starting latitude [°]
            ini.lat_0_a = DATBsample.ftr_LAT(1);
            % Starting longitude [°]
            ini.long_0_a = DATBsample.ftr_LON(1);
            % Starting altitude [m]
            ini.z0_a = DATBsample.ftr_alt(1);
            % Starting x position [m]
            ini.x_0 = (ini.lat_0_a - DATBsample.ftr_LAT(1)).*R_earth.*(pi/180);
            % Starting y position [m]
            ini.y_0 = (ini.long_0_a - DATBsample.ftr_LON(1)).*R_earth.*(pi/180);
            % Starting z velocity [m/s]
            ini.vz0_a = 0;
            % Starting x velocity [m/s]
```

```
        ini.vx0_a = 0;
        % Starting y velocity [m/s]
        ini.vy0_a = 0;
        % Starting volume [m^3]
        ini.V_0= 1.1;    % Initial Helium volume [m^3]
        ini0 = ini;      % Initial conditions at ground
        [simulation1] = segment(date, hour, ini, ini, DATBsample, wind, temp_model, np_He1);

        %% 2nd segment Simulation
        % Starting time [s]
        ini2.t0 = real_hour*3600;
        % Simulation time deviation [s]
        ini2.sim_time = slope_point;
        % Ascent parameters
        % Polytropic index
        np_He2 = np_He_2nd;
        % Starting latitude [°]
        ini2.lat_0_a = DATBsample.ftr_LAT(simulation1.time==ini2.t0);
        % Starting longitude [°]
        ini2.long_0_a = DATBsample.ftr_LON(simulation1.time==ini2.t0);
        % Starting altitude [m]
        z_a = simulation1.state(:,3);
        ini2.z0_a = z_a(simulation1.time==ini2.t0+ini2.sim_time);
        % Starting x position [m]
        x_0 = simulation1.state(:,1);
        ini2.x_0 = x_0(simulation1.time==ini2.t0+ini2.sim_time);
        % Starting y position [m]
        y_0 = simulation1.state(:,2);
        ini2.y_0 = y_0(simulation1.time==ini2.t0+ini2.sim_time);
        % Starting z velocity [m/s]
        vz_a = simulation1.state(:,6);
        ini2.vz0_a = vz_a(simulation1.time==ini2.t0+ini2.sim_time);
        % Starting x velocity [m/s]
        vx_a = simulation1.state(:,4);
        ini2.vx0_a = vx_a(simulation1.time==ini2.t0+ini2.sim_time);
        % Starting y velocity [m/s]
        vy_a = simulation1.state(:,5);
        ini2.vy0_a = vy_a(simulation1.time==ini2.t0+ini2.sim_time);
        % Starting volume [m^3]
        R = simulation1.R(ini2.sim_time);
        ini2.V_0 = (R^3)*(4/3)*pi;
        [simulation2] = segment(date, hour, ini2, ini0, DATBsample, wind, temp_model,
np_He2);
    end
  end
  simulation.time=[simulation1.time(1:ini2.sim_time);simulation2.time]-simulation1.time(1);
  simulation.state=[simulation1.state(1:ini2.sim_time,:);simulation2.state];
  simulation.R=[simulation1.R(1:ini2.sim_time) simulation2.R];
  simulation.v_wind=[simulation1.v_wind(1:ini2.sim_time,:);simulation2.v_wind];
  simulation.t_wind=[simulation1.t_wind(1:ini2.sim_time) simulation2.t_wind];
  %% Simulation postprocess plots and outputs
  [h, lat, lon] = postprocess_ascent(date,hour,ini,np_He1,np_He2,DATBsample,simulation);
  final_position = [h(end);lat(end);lon(end)];
  final_position_data =
[DATBsample.ftr_alt(end);DATBsample.ftr_LAT(end);DATBsample.ftr_LON(end)];
  final_toexcel = [final_position_data final_position];
  if save == 1
    cd ..; cd ..; cd Results;
```

```
    save(strcat(date,'simulation'),'simulation');
    cd ..; cd simulation;
end
if savemonth == 1
    cd ..; cd ..; cd Results;
    save(strcat(date(1:6),'simulation'),'simulation');
    cd ..; cd simulation;
end
```

B.8.1. Function segment

```
%segment Calculates the simulation for each fit which corresponds to
%certain polytropic index for each ascent segment.
%   Inputs:
%     date          :  Date of launch in format yyyymmdd
%     hour          :  Launch hour GMT+1 [h]
%     ini           :  Structure with segment initial conditions
%     ini0          :  Structure with launch initial conditions
%     DATBsample    :  Structure with information of launch registered
%     wind          :  Structure with wind data
%     temp_model    :  Temperature model used in simulation
%       1: Balloon Temperature = Ambient Temperature + ideal gas law
%       2: Polytropic model + ideal gas law
%       3: Polytropic or Isentropic/adiabatic model + ideal gas law
%     np_He         :  Structure with launch initial conditions
%   Outputs:
%     simulation    :  Structure containing simulation outputs

%% Real launch data
year = str2double(date(1:4));
month = str2double(date(5:6));
day = str2double(date(7:8));
if np_He == 1
   temp_model =1;
elseif np_He == 1.67
   temp_model =2;
else
end
if month ~=1
   dayOfYear = day + sum(eomday(year,1:(month-1)));
else
   dayOfYear = day;
end

%% Parameters
% Mass of payload [kg]
m_pay = 0.2;
% Mass of balloon [kg]
m_b = 0.35;
% Mass of parachute [kg]
m_p = 0.0; % Approximate value
% Mass of whole structure [kg]
m_s = m_pay+m_b+m_p;
% Initial Helium volume [m^3]
V_0 = ini0.V_0;
%     [T_b0, ~, P_b0, rho_b0, ~] =  AtmosUSSA(ini.z0_a);
[T_b0, ~, P_b0, ~, ~] = atmosNRL(ini0.z0_a, ini0.lat_0_a, ini0.long_0_a, year, dayOfYear,
ini0.t0);
% Burst Radius [m]
hmax = max(DATBsample.ftr_alt);
```

```
lat_hmax = DATBsample.ftr_LAT(DATBsample.ftr_alt==max(DATBsample.ftr_alt));
lon_hmax = DATBsample.ftr_LON(DATBsample.ftr_alt==max(DATBsample.ftr_alt));
time_hmax = ini.t0+DATBsample.ftr_time(DATBsample.ftr_alt==max(DATBsample.ftr_alt));
[T_burst, ~, P_burst, ~, ~] = atmosNRL(hmax, lat_hmax, lon_hmax, year, dayOfYear,
time_hmax);
V_burst = (P_b0*V_0/(8.314*T_b0))*8.314*(T_burst/P_burst);
R_burst = (0.75*V_burst/pi)^(1/3);

%% Plot experiment for live plot
% Altitude
if exist('DATBsample','var')==1
    figure
    % plot(ascend.time_alt,ascend.altitude)
    plot(DATBsample.ftr_time+ini.t0,DATBsample.ftr_alt)
    if temp_model ==1
        titlename = char(strcat('Isothermal Altitude vs time ',{' '},date, sprintf('%g',hour)));
    elseif  temp_model == 2
        titlename = char(strcat(sprintf('Polytropic %3g Altitude vs time ',np_He),{' '},date,
sprintf('%g',hour)));

    elseif temp_model == 3
        titlename = char(strcat('Adiabatic Altitude vs time ',{' '},date, sprintf('%g',hour)));
    end
    title(titlename)
    hold on
end
tic
%% Ascent Simulation
[time,state,R,v_wind_rel,t_wind_rel] = ...
    ascensionV2(ini,year,dayOfYear,temp_model,np_He,m_s,R_burst,wind,ini0);
toc

%% Outputs
simulation.time=time;
simulation.state=state;
simulation.R=R;
simulation.v_wind=v_wind_rel;
simulation.t_wind=t_wind_rel;
close all;
end
```

B.8.2. Function ascensionV2

```
function [t,state,Radius,v_wind_rel,t_wind_rel] = ascensionV2(varargin)
%ascensionV2 is a modification to function ascension, that calculates the
%balloon's trajectory in ascension, with the possibility of starting the
%simulation at a certain point of the trajectory.
%   Inputs:
%     ini          :  Structure with segment initial conditions
%     year         :  Year of launch
%     dayOfYear    :  Simulation day counting from 1st Jan
%     temp_model   :  Temperature model used in simulation
%       1: Balloon Temperature = Ambient Temperature + ideal gas law
%       2: Polytropic model + ideal gas law
%       3: Polytropic or Isentropic/adiabatic model + ideal gas law
%     np_He        :  Segment polytropic index
%     m_s          :  Mass of whole structure [kg]
%     R_burst      :  Burst Radius [m]
```

```
%    lat_0_a       :  Starting latitude [°]
%    lon_0_a       :  Starting longitude [°]
%    wind          :  Structure with wind data
%    ini0          :  Structure with launch initial conditions
%  Outputs:
%    time          :  Time vector of simulation [s]
%    state         :  Position and velocity in x-,y- and z-direction for
%                     the whole ascension [m]/[m/s]
%    Radius        :  Final radius [m]
%    v_wind_rel    :  Relative wind velocity vector [m/s]
%    t_wind_rel    :  Relative wind velocity time vector [s]

%% Data and initialisation
x0 = varargin{1}.x_0;          % Starting position [m]
y0 = varargin{1}.y_0;
z0 = varargin{1}.z0_a;
vx0 = varargin{1}.vx0_a;       % Starting velocity [m/s]
vy0 = varargin{1}.vy0_a;
vz0 = varargin{1}.vz0_a;
V_0 = varargin{1}.V_0;         % Initial Helium volume [m^3]
if vz0 == 0, vz0 = 1E-20; end
if vx0 == 0, vx0 = 1E-20; end
if vy0 == 0, vy0 = 1E-20; end
lat_0 = varargin{1}.lat_0_a;   % Starting latitude [°]
long_0 = varargin{1}.long_0_a; % Starting longitude [°]
t0 = varargin{1}.t0;           % Starting time [s]
sim_time = varargin{1}.sim_time;% Simulation deviation time [s]
year = varargin{2};
dayOfYear = varargin{3};
option = varargin{4};          % Temperature Model
np_He = varargin{5};           % Polytropic index
m_pay = varargin{6};           % Mass of payload [kg]
R_burst = varargin{7};         % Burst Radius [m]
wind_dat = varargin{8};        % Wind data [m/s]

z00 = varargin{9}.z0_a;        % Starting position [m]
lat_00 = varargin{9}.lat_0_a;  % Ground Starting latitude [°]
long_00 = varargin{9}.long_0_a;% Ground Starting longitude [°]

% Effective radius [m]
R0 = (V_0*(3/4)/pi).^(1/3);

% Earth radius [m]
R_earth = 6371;

% Initial ambient air parameters
% [T_a00, ~, p_a00, ~, ~] =  AtmosUSSA(z0);
[T_a00, ~, p_a00, ~, ~] = atmosNRL(z00, lat_00, long_00, year, dayOfYear, t0);
[T_a0, ~, p_a0, ~, ~] = atmosNRL(z0, lat_0, long_0, year, dayOfYear, t0);
% Initial balloon temperature equals initial ambient temperature [K]
T_b0 = T_a00;
T_bt = T_b0;

% Thermodynamical properties
R_universal = 8.314;       % Universal gas constant [kgm^2/s^2molK]
rho_He = 0.1685;           % Density of Helium [kg/m^3]
gamma_He = 1.666667;       % Heat capacity ratio He
Cp_He = 5190;              % Specific isobar heat capacity [J/KgK]
kappa_He = 0.149;          % Thermal conductivity [W/m^2K]
```

```
% Helium mass [kg]
n_He = p_a0*V_0/(R_universal*mean(T_a0));
m_He = 4e-3*n_He;

% Prepare wind data
% Pressure steps for wind data
p_wind = wind_dat{1,1}.wind.p;
p_wind_vert = wind_dat{1,1}.wind.p_vert;

% Grid
wind.grid_v = wind_dat{1,1}.wind.grid_v;
wind.grid_u = wind_dat{1,1}.wind.grid_u;

% Time steps for wind data
t_wind_steps = [0,6,12,18]*3600;

% Simulation
% Set initial radius
R = R0;

% Time Span
tspan = t0+sim_time:1:t0+sim_time+10000;
% Options for solver
options = odeset('Events',@ReachingTop,'RelTol',1e-10,'AbsTol',1e-3,...
    'OutputFcn',@odeplot,'OutputSel',3);
% options = odeset('Events',@ReachingTop,'RelTol',1e-10,'AbsTol',1e-3);
% Initial input

state0 = [x0;y0;z0;vx0;vy0;vz0];

% Count variables for event
ii = 1; yy = 0; xx = 0; zz = 0;

% Solving of the differential equation
[t,state] = ode45(@rise,tspan,state0,options);

    %% Differential equation of ascension
    function [X] = rise(t,state)
    % This function solves the differential equation for the ascension
    % for every time step

    % Position from input [m]
    x = state(1);
    y = state(2);
    z = state(3);
    % Velocity from input [m/s]
    vx = state(4);
    vy = state(5);
    vz_star = state(6);


    %% Wind velocity [m/s]
        % Actual time
        t_actual = t;
%         if t_actual > 40925%40925
%             warning('Point reached')
%         end
```

```
t_first = max(t_wind_steps((t_wind_steps<=t_actual)==1));
t_second = min(t_wind_steps((t_wind_steps>=t_actual)==1));
% Actual position in lat/long
if size(yy)>1
lat  = lat_0 + (yy(1:end)/R_earth)*(180/pi);
long = long_0 + (xx(1:end)/R_earth)*(180/pi)./cos(lat*pi/180);
else
   lat = lat_0; long = long_0;
end

if long < 0
   long = long+360;
end

% Get atmospheric properties from USSA and gravity
[g,~] = GeopotentialModel(z);
%    [T_a, ~, p_a, rho_a, mu_a] =  AtmosUSSA(z);
[T_a, ~, p_a, rho_a, mu_a] = atmosNRL(z, lat, long, year, dayOfYear, t_actual);
% Find corresponding values for actual pressure, lat and long
   % Pressure
   if p_wind(1)<p_a
      pos1_a = 1;
   else
      pos1_a = find(p_wind==min(p_wind((p_a<=p_wind)==1)));
   end
   if p_a < min(p_wind)
      pos1_b = pos1_a;
   else
      pos1_b = find(p_wind==max(p_wind((p_a>=p_wind)==1)));
   end
        if abs(p_wind(pos1_a)-p_a)<abs(p_wind(pos1_b)-p_a)
           pos1 = pos1_a;
        else
           pos1 = pos1_b;
        end

   % Latitude
   pos2_a = find(wind.grid_v{pos1}.lat==...
      min(wind.grid_v{pos1}.lat((lat<=...
      wind.grid_v{pos1}.lat)==1)));
   pos2_b = find(wind.grid_v{pos1}.lat==...
      max(wind.grid_v{pos1}.lat((lat>=...
      wind.grid_v{pos1}.lat)==1)));

   % Longitude
   pos3_a = find(wind.grid_v{pos1}.lon==...
      min(wind.grid_v{pos1}.lon((long<=...
      wind.grid_v{pos1}.lon)==1)));
   if long <=360 && long>= 359.5
      pos3_b = find(wind.grid_v{pos1}.lon==...
         min(wind.grid_v{pos1}.lon((long>=...
         wind.grid_v{pos1}.lon)==1)));
   else
      pos3_b = find(wind.grid_v{pos1}.lon==...
         min(wind.grid_v{pos1}.lon((long<=...
         wind.grid_v{pos1}.lon)==1)));
   end

   % Positions for vertical velocity pressure
```

```
    if p_wind_vert(1)<p_a
        pos4_a = 1;
    else
        pos4_a = find(p_wind_vert==min(p_wind_vert((p_a<=p_wind_vert)==1)));
    end
    if p_a < min(p_wind_vert)
        pos4_b = pos4_a;
    else
        pos4_b = find(p_wind_vert==max(p_wind_vert((p_a>=p_wind_vert)==1)));
    end

% Interpolation for the time
    for jj=1:2
        % Choosing wind corresponding to time
        if jj == 1
            if t_first == 0
                wind_x = wind_dat{1,1}.wind.wind_u;
                wind_y = wind_dat{1,1}.wind.wind_v;
                wind_z = wind_dat{1,1}.wind.wind_vert;
            elseif t_first == 6*3600
                wind_x = wind_dat{1,2}.wind.wind_u;
                wind_y = wind_dat{1,2}.wind.wind_v;
                wind_z = wind_dat{1,2}.wind.wind_vert;
            elseif t_first == 12*3600
                wind_x = wind_dat{1,3}.wind.wind_u;
                wind_y = wind_dat{1,3}.wind.wind_v;
                wind_z = wind_dat{1,3}.wind.wind_vert;
            elseif t_first == 18*3600
                wind_x = wind_dat{1,4}.wind.wind_u;
                wind_y = wind_dat{1,4}.wind.wind_v;
                wind_z = wind_dat{1,4}.wind.wind_vert;
            end
        elseif jj == 2
            if t_second == 0
                wind_x = wind_dat{1,1}.wind.wind_u;
                wind_y = wind_dat{1,1}.wind.wind_v;
                wind_z = wind_dat{1,1}.wind.wind_vert;
            elseif t_second == 6*3600
                wind_x = wind_dat{1,2}.wind.wind_u;
                wind_y = wind_dat{1,2}.wind.wind_v;
                wind_z = wind_dat{1,2}.wind.wind_vert;
            elseif t_second == 12*3600
                wind_x = wind_dat{1,3}.wind.wind_u;
                wind_y = wind_dat{1,3}.wind.wind_v;
                wind_z = wind_dat{1,3}.wind.wind_vert;
            elseif t_second == 18*3600
                wind_x = wind_dat{1,4}.wind.wind_u;
                wind_y = wind_dat{1,4}.wind.wind_v;
                wind_z = wind_dat{1,4}.wind.wind_vert;
            end
        end
% Corresponding wind velocities to altitude (interpolation between
% altitude, longitude and latitude (8 points))
    % X-direction (8 nearest points around actual point)
    v_wind_x_000 = wind_x{pos1_a}(pos2_a,pos3_a);
    v_wind_x_200 = wind_x{pos1_b}(pos2_a,pos3_a);
    v_wind_x_002 = wind_x{pos1_a}(pos2_a,pos3_b);
    v_wind_x_202 = wind_x{pos1_b}(pos2_a,pos3_b);
```

```
v_wind_x_020 = wind_x{pos1_a}(pos2_b,pos3_a);
v_wind_x_220 = wind_x{pos1_b}(pos2_b,pos3_a);
v_wind_x_022 = wind_x{pos1_a}(pos2_b,pos3_b);
v_wind_x_222 = wind_x{pos1_b}(pos2_b,pos3_b);
% X-direction (Alitude interpolated)
if pos1_a == pos1_b
   v_wind_x_100 = v_wind_x_000;
   v_wind_x_102 = v_wind_x_002;
   v_wind_x_120 = v_wind_x_020;
   v_wind_x_122 = v_wind_x_022;
else
   v_wind_x_100 = v_wind_x_000+((v_wind_x_200-v_wind_x_000)...
   /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
   v_wind_x_102 = v_wind_x_002+((v_wind_x_202-v_wind_x_002)...
   /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
   v_wind_x_120 = v_wind_x_020+((v_wind_x_220-v_wind_x_020)...
   /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
   v_wind_x_122 = v_wind_x_022+((v_wind_x_222-v_wind_x_022)...
   /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
end
% X-direction (Longitude interpolated)
if pos3_a == pos3_b
   v_wind_x_101 = v_wind_x_100;
   v_wind_x_121 = v_wind_x_120;
else
   v_wind_x_101 = v_wind_x_100+((v_wind_x_102-v_wind_x_100)...
      /(wind.grid_u{pos1}.lon(pos3_b)-...
      wind.grid_u{pos1}.lon(pos3_a)))*...
      (long-wind.grid_u{pos1}.lon(pos3_a));
   v_wind_x_121 = v_wind_x_120+((v_wind_x_122-v_wind_x_120)...
      /(wind.grid_u{pos1}.lon(pos3_b)-...
      wind.grid_u{pos1}.lon(pos3_a)))*...
      (long-wind.grid_u{pos1}.lon(pos3_a));
end
% X-direction (Latitude interpolated)
if pos2_a == pos2_b
   v_wind_x(jj) = v_wind_x_101;
else
   v_wind_x(jj) = v_wind_x_101+((v_wind_x_121-v_wind_x_101)...
      /(wind.grid_u{pos1}.lat(pos2_b)-...
      wind.grid_u{pos1}.lat(pos2_a)))*...
      (lat-wind.grid_u{pos1}.lat(pos2_a));
end

% Y-direction (8 nearest points around actual point)
v_wind_y_000 = wind_y{pos1_a}(pos2_a,pos3_a);
v_wind_y_200 = wind_y{pos1_b}(pos2_a,pos3_a);
v_wind_y_002 = wind_y{pos1_a}(pos2_a,pos3_b);
v_wind_y_202 = wind_y{pos1_b}(pos2_a,pos3_b);
v_wind_y_020 = wind_y{pos1_a}(pos2_b,pos3_a);
v_wind_y_220 = wind_y{pos1_b}(pos2_b,pos3_a);
v_wind_y_022 = wind_y{pos1_a}(pos2_b,pos3_b);
v_wind_y_222 = wind_y{pos1_b}(pos2_b,pos3_b);
% Y-direction (Alitude interpolated)
if pos1_a == pos1_b
   v_wind_y_100 = v_wind_y_000;
   v_wind_y_102 = v_wind_y_002;
   v_wind_y_120 = v_wind_y_020;
   v_wind_y_122 = v_wind_y_022;
```

```
else
    v_wind_y_100 = v_wind_y_000+((v_wind_y_200-v_wind_y_000)...
    /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
    v_wind_y_102 = v_wind_y_002+((v_wind_y_202-v_wind_y_002)...
    /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
    v_wind_y_120 = v_wind_y_020+((v_wind_y_220-v_wind_y_020)...
    /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
    v_wind_y_122 = v_wind_y_022+((v_wind_y_222-v_wind_y_022)...
    /(p_wind(pos1_b)-p_wind(pos1_a)))*(p_a-p_wind(pos1_a));
end

% Y-direction (Longitude interpolated)
if pos3_a == pos3_b
    v_wind_y_101 = v_wind_y_100;
    v_wind_y_121 = v_wind_y_120;
else
    v_wind_y_101 = v_wind_y_100+((v_wind_y_102-v_wind_y_100)...
        /(wind.grid_u{pos1}.lon(pos3_b)-...
        wind.grid_u{pos1}.lon(pos3_a)))*...
        (long-wind.grid_u{pos1}.lon(pos3_a));
    v_wind_y_121 = v_wind_y_120+((v_wind_y_122-v_wind_y_120)...
        /(wind.grid_u{pos1}.lon(pos3_b)-...
        wind.grid_u{pos1}.lon(pos3_a)))*...
        (long-wind.grid_u{pos1}.lon(pos3_a));
end
% Y-direction (Latitude interpolated)
if pos2_a == pos2_b
    v_wind_y(jj) = v_wind_y_101;
else
    v_wind_y(jj) = v_wind_y_101+((v_wind_y_121-v_wind_y_101)...
        /(wind.grid_u{pos1}.lat(pos2_b)-...
        wind.grid_u{pos1}.lat(pos2_a)))*...
        (lat-wind.grid_u{pos1}.lat(pos2_a));
end

% Z-direction (8 nearest points around actual point)
v_wind_z_000 = wind_z{pos4_a}(pos2_a,pos3_a);
v_wind_z_200 = wind_z{pos4_b}(pos2_a,pos3_a);
v_wind_z_002 = wind_z{pos4_a}(pos2_a,pos3_b);
v_wind_z_202 = wind_z{pos4_b}(pos2_a,pos3_b);
v_wind_z_020 = wind_z{pos4_a}(pos2_b,pos3_a);
v_wind_z_220 = wind_z{pos4_b}(pos2_b,pos3_a);
v_wind_z_022 = wind_z{pos4_a}(pos2_b,pos3_b);
v_wind_z_222 = wind_z{pos4_b}(pos2_b,pos3_b);
% Z-direction (Alitude interpolated)
if pos4_a == pos4_b
    v_wind_z_100 = v_wind_z_000;
    v_wind_z_102 = v_wind_z_002;
    v_wind_z_120 = v_wind_z_020;
    v_wind_z_122 = v_wind_z_022;
else
    v_wind_z_100 = v_wind_z_000+((v_wind_z_200-v_wind_z_000)...
    /(p_wind_vert(pos4_b)-p_wind_vert(pos4_a)))*(p_a-p_wind_vert(pos4_a));
    v_wind_z_102 = v_wind_z_002+((v_wind_z_202-v_wind_z_002)...
    /(p_wind_vert(pos4_b)-p_wind_vert(pos4_a)))*(p_a-p_wind_vert(pos4_a));
    v_wind_z_120 = v_wind_z_020+((v_wind_z_220-v_wind_z_020)...
    /(p_wind_vert(pos4_b)-p_wind_vert(pos4_a)))*(p_a-p_wind_vert(pos4_a));
    v_wind_z_122 = v_wind_z_022+((v_wind_z_222-v_wind_z_022)...
```

```
            /(p_wind_vert(pos4_b)-p_wind_vert(pos4_a)))*(p_a-p_wind_vert(pos4_a));
        end
        % Z-direction (Longitude interpolated)
        if pos3_a == pos3_b
            v_wind_z_101 = v_wind_z_100;
            v_wind_z_121 = v_wind_z_120;
        else
            v_wind_z_101 = v_wind_z_100+((v_wind_z_102-v_wind_z_100)...
                /(wind.grid_u{pos1}.lon(pos3_b)-...
                wind.grid_u{pos1}.lon(pos3_a)))*...
                (long-wind.grid_u{pos1}.lon(pos3_a));
            v_wind_z_121 = v_wind_z_120+((v_wind_z_122-v_wind_z_120)...
                /(wind.grid_u{pos1}.lon(pos3_b)-...
                wind.grid_u{pos1}.lon(pos3_a)))*...
                (long-wind.grid_u{pos1}.lon(pos3_a));
        end
        % Z-direction (Latitude interpolated)
        if pos2_a == pos2_b
            v_wind_z(jj) = v_wind_z_101;
        else
            v_wind_z(jj) = v_wind_z_101+((v_wind_z_121-v_wind_z_101)...
                /(wind.grid_u{pos1}.lat(pos2_b)-...
                wind.grid_u{pos1}.lat(pos2_a)))*...
                (lat-wind.grid_u{pos1}.lat(pos2_a));
        end
        end
if v_wind_x(1) == v_wind_x(2)
    v_wind_x = v_wind_x(1);
else
   v_wind_x = v_wind_x(1)+((v_wind_x(2)-v_wind_x(1))...
        /(t_second-t_first))*(t_actual-t_first);
end
if v_wind_y(1) == v_wind_y(2)
    v_wind_y = v_wind_y(1);
else
  v_wind_y = v_wind_y(1)+((v_wind_y(2)-v_wind_y(1))...
        /(t_second-t_first))*(t_second-t_actual);
end
if v_wind_z(1) == v_wind_z(2)
    v_wind_z = v_wind_z(1);
else
   v_wind_z = v_wind_z(1)+((v_wind_z(2)-v_wind_z(1))...
        /(t_second-t_first))*(t_second-t_actual);
end

%% Balloon Temperature and Radius depending on model
if option == 1
    T_b = T_a;
elseif option == 2
   % Polytropic
   T_b = power((p_a/p_a00),((np_He-1)/np_He))*T_bt;
elseif option ==3
   % Isentropic / Adiabatic
   T_b = power((p_a/p_a00),((gamma_He-1)/gamma_He))*T_bt;
end
V = n_He*R_universal*T_b./p_a;
R = (V*(3/4)/pi).^(1/3);
%% Calculate total mass [kg]
% Effective area [m²]
```

```
A = pi*R^2;
% Virtual mass [kg]
m_a = rho_a * V;
% Total balloon mass [kg]
m_tot = m_He+m_pay;

%% Get Cd
Re = (rho_a*R*vz_star)/mu_a;

% Drag coefficient [1]
Cd = 0.04808*(log(Re))^2-1.406*log(Re)+10.49;
Cd = real(Cd);

%% Calculate acceleration forces
F_Drag_x = 0.5*Cd*A*rho_a*state(4)*abs(state(4)); % Drag Force x
F_Drag_y = 0.5*Cd*A*rho_a*state(5)*abs(state(5)); % Drag Force y
F_Drag_z = 0.5*Cd*A*rho_a*state(6)*abs(state(6)); % Drag Force z
F_Lift = (m_a - m_tot)*g;          % Lift Force

% Wind forces
F_wind_x = 0.5*rho_a*v_wind_x*abs(v_wind_x)*A;
F_wind_y = 0.5*rho_a*v_wind_y*abs(v_wind_y)*A;
F_wind_z = 0.5*rho_a*v_wind_z*abs(v_wind_z)*A;

% Acceleration [m/s²]
dvx = (F_wind_x-F_Drag_x)/m_tot;
dvy = (F_wind_y-F_Drag_y)/m_tot;
dvz = (F_Lift-F_Drag_z+F_wind_z)/m_tot;

%% Define output (velocity [m/s] and acceleration [m/s²])
X = [state(4);state(5);state(6);dvx;dvy;dvz]; X = double(X);

% Relative wind speed
v_wind_rel(ii,1:3) = [state(4)-v_wind_x;state(5)-v_wind_y;...
    state(6)-v_wind_z];
t_wind_rel(ii) = t;

% Parameters of step n stored for step n+1
p_a00 = p_a;
T_bt = T_b;

end

%% Event Function to stop ode45
function [value,isterminal,direction] = ReachingTop(t,state)
% This function serves as event to stop the differential equation
% solver and therefore the simulation
xx(ii) = state(1); yy(ii) = state(2); zz(ii) = state(3); Radius(ii) = R;

% End the simulation, when balloon reaches highest point on the
% trajectory (when it "stops" to rise) or the balloon bursts
if ii>20
    % End Condition
    value = double(mean(zz(ii-20:ii)-zz(ii-20)) > 5E-1 & R_burst >= R);
else
    value = 1;
end
isterminal = 1;     % End calculation after condition is met
```

```
    direction = 0;      % Local Minimum/Maximum
    ii = ii+1;
    end

end
```

B.8.3. Function postprocess_ascent

```
function [altitude_ascent,lat_ascent, long_ascent ] = postprocess_ascent(date,
hour,ini,np_He1,np_He2,DATBsample,simulation)
%findfit Calculates the simulation for each fit which corresponds to
%certain polytropic index for each ascent segment.
%   Inputs:
%     date          :  Date of launch in format yyyymmdd
%     hour          :  Launch hour GMT+1 [h]
%     ini           :  Structure with segment initial conditions
%     DATBsample    :  Structure with information of launch registered
%     during the defined year, month, day and hour
%     simulation    :  Structure containing simulation outputs
%   Outputs:
%     altitude_ascent: Simulation end altitude
%     lat_ascent    :  Simulation end latitude
%     long_ascent   :  Simulation end longitude
%     Altitude, Vertical velocity, flight path and relative wind vel plots

%% Post processing ascent
subplotmode =1;
% Earth radius [m]
R_earth = 6371000;
% Altitude [m]
altitude_ascent = simulation.state(:,3);
% Vertical velocity [m/s]
vvel_ascent = diff(altitude_ascent)./diff(simulation.time);
time_vvel_ascent = simulation.time(1:end-1)+diff(simulation.time(:))/2;
% Getting latitudes and longitudes for distances in x and y
% x- and y-coordinates
x_ascent = simulation.state(:,1);
y_ascent = simulation.state(:,2);
% Transform to lat, long
lat_ascent  = ini.lat_0_a + (y_ascent(1:end-1)./R_earth).*(180/pi);
long_ascent = ini.long_0_a + (x_ascent(1:end-1)/R_earth)*(180/pi)./cos(lat_ascent*pi/180);
% Altitude for latitude and longitude
alt_latlong_ascent = altitude_ascent(1:end-1)+diff(altitude_ascent(1:end))/2;

%% Plot ascent
cd ..; cd Results;
%% Altitude
if subplotmode==1
    figure(2)
    subplot(2,2,1)
    pos = get(gca, 'Position');
    pos(1) = 0.035;
    pos(3) = pos(1)+0.435;
    set(gca, 'Position', pos)
else
    figure(2)
end
    plot(simulation.time,altitude_ascent)
hold on
if exist('DATBsample','var')==1
```

```
    plot(DATBsample.ftr_time ,DATBsample.ftr_alt)
    legend('Simulation',['Data (',date,')'],'Location','best')%,'FontSize',16)
else
    legend('Simulation','Location','best')%,'FontSize',16)
end
titlename = char(strcat(date, sprintf('%g',hour), ' Altitude vs time segment n1 = ',
sprintf('%1.3f',np_He1), ' n2 = ', sprintf('%1.3f',np_He2)));
title(titlename)%,'FontSize',16)
xlabel('time [s]')%,'FontSize',16)
ylabel('Altitude [m]')%,'FontSize',16)

%% Velocity
if subplotmode==1
    subplot(2,2,2)
    pos = get(gca, 'Position');
    pos(1) = 0.535;
    pos(3) = 0.4425;
    set(gca, 'Position', pos)
else
    saveas(figure(2),strcat(titlename,'.fig'))
    saveas(figure(2),strcat(titlename,'.png'))
    figure(3)
end
plot(time_vvel_ascent,vvel_ascent)
hold on
if exist('DATBsample','var')==1
    plot(DATBsample.ftr_time,DATBsample.calc_Vvert)
    legend('Simulation',['Data (',date,')'],'Location','best')
else
    legend('Simulation','Location','best')
end
titlename = char(strcat(date, sprintf('%g',hour), ' Vert vel vs time segment n1 = ',
sprintf('%1.3f',np_He1), ' n2 = ', sprintf('%1.3f',np_He2)));
title(titlename)
xlabel('time [s]')
ylabel('Vertical velocity [m/s]')

%% Flight path (trajectory)
if subplotmode==1
    subplot(2,2,3)
    pos = get(gca, 'Position');
    pos(1) = 0.035;
    pos(3) = pos(1)+0.435;
    set(gca, 'Position', pos)
else
    saveas(figure(3),strcat(titlename,'.fig'))
    saveas(figure(3),strcat(titlename,'.png'))
    figure(4)
end
if exist('DATBsample','var')==1
    plot3(DATBsample.ftr_LON,DATBsample.ftr_LAT,DATBsample.ftr_alt,'blue');    hold on
end
plot3(long_ascent,lat_ascent,alt_latlong_ascent,'green');   hold on
scatter3(long_ascent(1),lat_ascent(1),alt_latlong_ascent(1),'green','filled',...
    'MarkerEdgeColor',[0 0 0]); hold on
scatter3(long_ascent(end),lat_ascent(end),altitude_ascent(end),'cyan',...
    'filled','MarkerEdgeColor',[0 0 0]); hold on
if exist('DATBsample','var')==1
```

```
    scatter3(DATBsample.ftr_LON(end),DATBsample.ftr_LAT(end),DATBsample.ftr_alt(end),...
        'cyan','filled','MarkerEdgeColor',[0 0 0]);
end
xlabel('Longitude [°]'); ylabel('Latitude [°]'); zlabel('Altitude [m]');
titlename = char(strcat(date, sprintf('%g',hour), ' Flight trajectory vs time segment n1 = ',
sprintf('%1.3f',np_He1), ' n2 = ', sprintf('%1.3f',np_He2)));
title(titlename)
if exist('DATBsample','var')==1
    legend('Data (Ascension)','Simulation (Ascension)', 'Launch site',...
        'Balloon burst (Simulation)','Balloon burst (Data)')
else
    legend('Simulation (Ascension)','Launch site',...
        'Balloon burst (Simulation)')
end

%% Relative wind velocity
if subplotmode==1
    subplot(2,2,4)
    pos = get(gca, 'Position');
    pos(1) = 0.535;
    pos(3) = 0.4425;
    set(gca, 'Position', pos)
else
    saveas(figure(4),strcat(titlename,'.fig'))
    saveas(figure(4),strcat(titlename,'.png'))
    figure(5)
end
plot(simulation.t_wind,simulation.v_wind(:,1),simulation.t_wind,simulation.v_wind(:,2),...
    simulation.t_wind,simulation.v_wind(:,3))
xlabel('time [s]')
ylabel('Relative velocity [m/s]')
legend('Relative wind in x-direction','Relative wind in y-direction',...
    'Relative wind in z-direction','Location','best')
titlename = char(strcat(date, sprintf('%g',hour), ' Relative 3D velocity segment n1 = ',
sprintf('%1.3f',np_He1), ' n2 = ', sprintf('%1.3f',np_He2)));
title(titlename)
if subplotmode==1
    % Enlarge figure to full screen.
    set(gcf, 'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);
    titlename = (char(strcat(date, sprintf('%g',hour), ' Segment n1 = ', sprintf('%1.3f',np_He1), ' n2
= ', sprintf('%1.3f',np_He2))));
    saveas(figure(2),strcat(titlename,'.fig'))
    saveas(figure(2),strcat(titlename,'.png'))
else
    saveas(figure(5),strcat(titlename,'.fig'))
    saveas(figure(5),strcat(titlename,'.png'))
end
close all;
cd ..; cd simulation;
end
```